International Journal of Industrial Engineering, 33(2), 218-239, 2026

A COMPARATIVE STUDY OF METAHEURISTIC ALGORITHMS FOR SCHEDULING ON UNRELATED PARALLEL MACHINES: MINIMIZING WEIGHTED EARLINESS–TARDINESS WITH NON-ZERO RELEASE TIMES AND DISTINCT DUE DATES

Alzira Mota[1, 3, *], Paulo Ávila[2, 3], Luís Afonso[1], João Bastos[2, 3], and Goran Putnik[4]

[1]Department of Mathematics
ISEP, Polytechnic of Porto
Porto, Portugal
[*]Corresponding author's e-mail: atm@isep.ipp.pt

[2]Department of Mechanical Engineering
ISEP, Polytechnic of Porto
Porto, Portugal

[3]Institute for Systems and Computer Engineering
Technology and Science (INESC TEC)
Porto, Portugal

[4]Algoritmi Research Center/LASI
University of Minho
Guimarães, Portugal

This study addresses the unrelated parallel machine scheduling problem in a just-in-time manufacturing context, aiming to minimize total weighted earliness and tardiness. The problem formulation incorporates non-zero release times and distinct due dates, reflecting realistic industrial environments. Three hybrid metaheuristic approaches: Genetic Algorithm, Tabu Search, and Variable Neighborhood Search, are proposed and analyzed. The main contribution of this work lies in integrating a linear-programming-based decoding procedure into each metaheuristic to determine job start times and accurately evaluate solution quality, while preserving the general structure of the unrelated parallel machine scheduling problem. The proposed methods are evaluated using a set of medium- and large-scale instances generated for this study. Computational analysis reveals differences in performance among the metaheuristics, with Tabu Search exhibiting the most consistent and effective behavior in terms of solution quality and convergence speed.

Keywords: Unrelated Parallel Machine Scheduling; Just-in-Time Scheduling; Total Weighted Earliness and Tardiness; Hybrid Metaheuristics.

(Received on July 7, 2025; Accepted on February 1, 2026)

1. INTRODUCTION

To remain competitive in a globalized, highly demanding market, manufacturing companies must adopt efficient production systems capable of meeting client deadlines with precision. In this context, the just-in-time (JIT) production strategy plays a central role in minimizing waste and optimizing resource allocation (Sugimori *et al.*, 1977; Rabbani *et al.*, 2018). While some authors have also associated JIT practices with broader organizational perspectives, such as governance and sustainability (Putnik and Ávila, 2016; Palhau *et al.*, 2024), this work considers these aspects only in a broader contextual sense.

A key operational challenge in JIT-oriented environments is ensuring that jobs are completed as close as possible to their due dates. Early completions may impose additional inventory or quality-degradation costs, whereas tardy completions may result in contractual penalties, reduced service levels, or lost sales. Weighted earliness-tardiness (ET) formulations explicitly address timing deviations by defining due dates as parameters or decision variables and may include symmetric or asymmetric penalties. Recent evidence on productivity–quality trade-offs (Grieco *et al.*, 2022) reinforces the practical relevance of asymmetric ET models, particularly in manufacturing settings where processing conditions or material variability influence both timing and cost.

This study focuses on the unrelated parallel machine scheduling problem (UPMSP) involving total weighted earliness–tardiness (TWET) minimization, distinct due dates, and non-zero release times. This formulation captures realistic scenarios encountered in manufacturing environments, including preventive maintenance, multi-line order fulfillment, and machine-dependent processing times with eligibility constraints. The problem generalizes well-known NP-hard variants such as scheduling on identical machines with common due dates (Cheng, 1989) and remains strongly NP-hard.

The main objective of this study is to develop and evaluate metaheuristic approaches, namely a Genetic Algorithm (GA), Tabu Search (TS), and Variable Neighborhood Search (VNS), supported by a linear programming (LP) model for optimal start-time determination. To the best of our knowledge, no prior work has simultaneously considered unrelated machines, distinct due dates, non-zero release times, and asymmetric ET penalties within a unified TWET minimization framework. This combination introduces an additional layer of complexity and realism to the problem, motivating the development of tailored optimization strategies.

The remainder of this paper is structured as follows: Section 2 presents a detailed literature review. Section 3 outlines the research methodology adopted in this study. Section 4 describes the scheduling problem under consideration. Section 5 formulates the linear model embedded within the metaheuristics. Section 6 details the proposed metaheuristic algorithms. Section 7 reports and discusses computational experiments and results. Finally, Section 8 concludes the paper and outlines directions for future research.

2. LITERATURE REVIEW

Earliness–tardiness scheduling has received extensive attention in both single- and parallel-machine environments. In parallel machine settings, machines are categorized as identical, uniform, or unrelated, depending on their processing characteristics. Numerous objectives have been studied, including mean absolute deviation, weighted deviations from due dates, and quadratic ET values. However, the present review focuses exclusively on studies that minimize the total weighted earliness–tardiness, expressed as a linear combination of earliness and tardiness penalties. This formulation is frequently adopted in practice due to its interpretability and linear structure.

Several survey papers summarise key foundational contributions to ET-based scheduling. Rolim and Nagano (2020) provide a comprehensive review of parallel-machine scheduling problems with common due dates, whereas Janiak *et al.* (2015) examine models based on due windows and their assignments. Kramer and Subramanian (2019) supply an annotated bibliography covering ET scheduling, relevant objective functions, and standard solution approaches. The review that follows is organized thematically, beginning with studies on identical-machine environments, then transitioning to contributions addressing unrelated parallel machines, and finally to recent extensions and TWET-specific developments.

In the case of identical machines, Chen and Powell (1999) formulated the TWET-identical parallel machine as an integer programming model and reformulated it as a set partitioning problem with side constraints using the Dantzig-Wolfe decomposition. They obtained an exact solution using a branch-and-bound algorithm that solved instances with up to 60 jobs. Sun and Wang (2003) analyzed proportional ET penalties with common due dates, proved NP-hardness, and proposed a dynamic programming approach and two list scheduling (LS) heuristics. Rios-Solis and Sourd (2008) addressed the restrictive common due date problem using pseudo-polynomial algorithms. More recently, T'kindt *et al.* (2020) presented a hybrid algorithm combining dynamic programming with a sort-and-search strategy for problems with symmetric lead-time weights and a non-restrictive common due date. This work was among the first to provide theoretical results on worst-case complexity for scheduling problems with non-regular criteria.

More scalable approaches for identical machines have relied on metaheuristics. The Fast Ruin and Recreate (FR&R) method by Lin *et al.* (2016), a hybrid genetic algorithm with iterated local search proposed by Amorim *et al.* (2017), and hybrid frameworks combining path relinking, local search, and exact subproblem evaluation (Alvarez-Valdes *et al.*, 2015).

The UPMSP introduces additional complexity because job processing times depend on the assigned machine (Ekici *et al.*, 2019; Pinedo, 2022). Bank and Werner (2001) considered different release dates and a common due date, explored the problem's structural properties, and proposed constructive approaches based on job allocation followed by per-machine scheduling. Plateau and Rios-Solis (2010) developed quadratic programming models for restrictive and non-restrictive common due dates, although their restrictive formulation was later improved by Beyranvand *et al.* (2012). M'hallah and Al-Khamis (2012) presented a mixed-integer linear programming (MILP) model and metaheuristic solutions, including tabu search and simulated annealing, for problems with distinct due dates but without release times. Polyakovskiy and M'Hallah (2014) introduced a multi-agent system (MAS^H) to improve workload distribution in UPMSP environments. Cheng and Huang (2017) proposed a modified genetic algorithm (GARTC) that jointly optimizes release times and schedules, achieving competitive results. Arik (2020) assessed the performance of the artificial bee colony (ABC), genetic algorithm, and simulated annealing algorithms, showing that ABC outperformed the alternatives in terms of solution quality. A related study by Arık *et al.* (2022) developed constructive heuristics and local search mechanisms for the UPMSP with a restrictive common due date, leveraging the V-shaped property to optimize machine start times and eliminate idle periods.

Table 1 summarises the main contributions addressing TWET minimization in parallel machine environments without additional modeling constraints. The table highlights key problem characteristics, such as release dates, due dates, weights, and machine configurations, as well as the solution methods employed. In this studies the objective function is expressed by $\sum_{i=1}^{n}(\alpha_i E_i + \beta_i T_i)$, where n is the number of jobs, α_i and β_i are the earliness and tardiness weights for the job i, and E_i and T_i denote the respective earliness and tardiness values. For clarity, all abbreviations used in the table are listed at the end.

Table 1. Summary of TWET Minimization Studies in Parallel Machine Scheduling

Authors	Release Date	Due Date	Weights	Environment	Method
Chen and Powell (1999)	NC	C	S	I	DA
Bank and Werner (2001)	C	C	A	U	CA
Sun and Wang (2003)	NC	C	A	I	DP/IH
Rios-Solis and Sourd (2008)	NC	C	A	I	DP
Plateau and Rios-Solis (2010)	NC	C	A	U	QP
Beyranvand *et al.* (2012)	NC	C	A	U	QP
M'hallah and Al-Khamis, (2012)	NC	D	A	U	TS/SA
Polyakovskiy and M'Hallah (2014)	NC	D	A	U	MAS
Alvarez-Valdes *et al.* (2015)	NC	C	A	I	PR/ss
Lin *et al.* (2016)	NC	C	A	I	FR&R
Amorim *et al.* (2017)	NC	D	A	I	HGA/ILS
Cheng and Huang (2017)	NC	D	S	U	MGA
T'kindt *et al.* (2020)	NC	C	S	I	DP
Arik (2020)	NC	C	S	U	ABC/GA/SA
Arık *et al.* (2022)	NC	C	A	U	CH/IH
Our study	C	D	A	U	GA/TA/VNS

Release Date C = Release constraint, NC = No release constraints
Due Date C = Common due dates, D = Distinct due dates
Weights S = Symmetric weights, A = Asymmetric weights
Environment I = Identical machines, U = Unrelated machines
Method DA = Decomposition Algorithm, CA: Constructive Algorithm, DP = Dynamic Programming, IH = Improvement Heuristic, QP = Quadratic Programming, TS = Tabu Search, SA = Simulated Annealing, MAS = Multi-Agent System, PR = Path Relinking, SS = Scatter Search, FR&R = Fast Ruin and Recreate, HGA = Hybrid Genetic Algorithm, ILS = Iterative Local Search, MGA = Modified Genetic Algorithm, GA = Genetic Algorithm, ABC = Artificial Bee Colony, CH = Constructive Heuristic, VNS = Variable Neighborhood Search.

In addition to distinguishing between identical and unrelated machine environments, several studies have introduced additional layers of complexity into the scheduling problem, including common due-date assignment problems, multi-objective formulations, learning and deterioration effects, and maintenance considerations.

Kim *et al.* (2012) examined the integration of common due date assignment into parallel machine scheduling. Kayvanfar *et al.* (2014) proposed a MILP model and heuristic approaches for the UPMSP to minimize total weighted tardiness, earliness, and job compression/expansion costs under makespan constraints. Zarandi and Kayvanfar (2015) introduced a bi-objective model for identical parallel machines with controllable processing time, balancing early/tardy penalties and makespan. This work was later extended by Aalaei *et al.* (2017) to incorporate job preemption and maintenance costs. Asghari and Nezhadali (2014) and Mota *et al.* (2025) studied a multi-objective scheduling problem that minimizes TWET and total flow time under fuzzy parameters and machine deterioration effects. Ekici *et al.* (2019) addressed TWET minimization with sequence-dependent setups, heterogeneous release dates, job–machine compatibility restrictions, and load-balancing considerations. Rezaeian *et al.* (2020) explored a fuzzy scheduling problem on unrelated parallel machines within a JIT production context.

More recently, Foroutan *et al.* (2024) addressed an unrelated parallel machine environment involving sequence-dependent family setups, machine eligibility constraints, and soft time windows. They aimed to minimize the total weighted earliness–tardiness while maximizing the number of just-in-time jobs. Their study compared simulated annealing, artificial immune systems, genetic algorithm, and ant colony optimization, with the latter, enhanced by a repair-based local search, achieving the best performance. Despite these advances, their assumption of a common due date introduces limitations when modeling systems with heterogeneous or job-specific timing requirements.

Zhang *et al.* (2025) examined parallel machine scheduling and due-date assignment under uncertain processing times through a distribution-free model based on mean–variance information. They established an extended smallest-variance-first rule as optimal for identical machines and introduced a mixed 0–1 second-order conic programming formulation with a two-phase approximation algorithm for unrelated machines. Although uncertainty is not considered in this study, their work reflects the growing interest in robust and realistic scheduling formulations.

Mor *et al.* (2025) studied parallel machine scheduling with machine-dependent generalized due dates. Their goals included total tardiness, maximum tardiness, the number of tardy jobs, and total late work. They demonstrated that all variants are NP-hard even when due-date structures are the same, but remain solvable in pseudo-polynomial time. Concerning the total late work criterion, they also showed fixed-parameter tractability with respect to the combined variability of due dates and processing times. These findings broaden the scope of due-date modeling in parallel machine scheduling and suggest possible advancements for unrelated machine environments.

Caricato *et al.* (2021) examined a parallel machine environment with sequence-dependent setups and workforce constraints, showing that operator availability and skill requirements can affect machine schedules. Snauwaert and Vanhoucke (2025) expanded this research by analyzing multi-skilled settings with hierarchical skill levels and illustrating their influence on scheduling choices. Together, these studies emphasize the increasing importance of workforce-aware approaches and propose methods to integrate operator skills and resource diversity into future extensions of the UPMSP.

Babu and Girish (2025) developed Pareto-optimal metaheuristics for jointly minimizing TWET and makespan on single machines with sequence-dependent setups. This confirms the ongoing relevance of TWET-based objectives, though not for multi-machine or unrelated environments.

These contributions demonstrate the variety of approaches to parallel machine scheduling across different configurations and modeling assumptions. However, there is a lack of studies that simultaneously consider distinct due dates, non-zero release times, and TWET minimization in the unrelated parallel machine environment. To the best of our knowledge, studies have only examined subsets of these characteristics, such as unrelated machines with common due dates or identical machines with distinct due dates and release constraints. Their formulations do not readily extend to settings in which job-dependent timing and heterogeneous machine capabilities must be addressed simultaneously. This gap is especially relevant in modern production systems, where variability in job availability and machine performance is common, and where both early and tardy completions incur asymmetric costs that must be balanced.

Therefore, it is essential to address this combination of constraints to develop scheduling models that more accurately reflect dynamic, heterogeneous manufacturing environments. These models can enable operationally realistic decision-making and support enhancements in service levels, cost management, and resource coordination by capturing both machine-dependent processing effects and job-specific temporal needs.

Motivated by this need, this study proposes metaheuristic strategies, namely, genetic algorithms, tabu search, and variable neighborhood search, that incorporate a linear programming model to compute the optimal job start times. The neighborhood structure includes inter- and intra-machine swaps, job insertion and movement operators, as well as two mechanisms, balance load and chain swap, designed to enhance search diversification and intensification.

3. RESEARCH METHODOLOGY

We evaluate the performance of three hybrid metaheuristic strategies using a quantitative, simulation-based experimental approach: Genetic Algorithms, Tabu Search, and Variable Neighborhood Search, applied to the UPMSP with total weighted earliness and tardiness objectives. Artificial benchmark instances are generated to represent production settings involving heterogeneous release times and distinct due dates.

All solution methods are implemented within a common framework. An LP model embedded within the metaheuristics acts as a decoder, computing the optimal start times for each candidate job schedule and the associated objective function value, which serves as the fitness measure during the search.

All algorithms are executed under controlled experimental conditions, and their performance is evaluated based on multiple independent runs, assessing robustness, convergence behavior, and solution quality. Implementation and execution details are provided in Section 7.1.

4. PROBLEM DESCRIPTION

We consider a set of n independent jobs, $J = \{J_1, J_2, \dots, J_n\}$, each associated with earliness weights $E = \{\alpha_1, \alpha_2, \dots, \alpha_n\}$, tardiness weights $T = \{\beta_1, \beta_2, \dots, \beta_n\}$, release dates $R = \{r_1, r_2, \dots, r_n\}$, and due dates $D = \{d_1, d_2, \dots, d_n\}$. A set of m unrelated parallel machines $M = \{1,2, \dots, m\}$ is available, each capable of processing any job. Each job J_i requires exactly one operation on a machine. Since the scheduling environment consists of unrelated parallel machines, the processing time of job J_i on machine k may vary and is denoted by p_{ik}. The indices i and j indicate jobs, while k refers to machines.

The following standard assumptions are adopted:

(1) All machines are available from time zero.
(2) Each machine processes at most one job at a time.
(3) Idle time on the machines is allowed to align job completion times with their due dates, accounting for release dates and the asymmetric cost of completing jobs either too early or too late.

(4) Jobs become available only at their release dates.
(5) Once started, processing cannot be interrupted (non-preemptive).
(6) Setup times are included in the processing times.

Each job has a specific due date d_i. Its earliness is computed according to Equation (1), which measures how much earlier than the due date the job finishes. If the completion time C_i occurs before d_i, the difference $d_i - C_i$ represents the earliness; otherwise, the value is zero.

$$E_i = max(d_i - C_i, 0), i \in J \tag{1}$$

Similarly, tardiness is defined in Equation (2) as the amount by which the job exceeds its due date. If the completion time is later than d_i, the term $C_i - d_i$ captures the delay; otherwise, the tardiness is zero.

$$T_i = max(C_i - d_i, 0), i \in J \tag{2}$$

Let Z denote the encoding vector representing the sequence of jobs assigned to each machine. The objective function in Equation (3) minimizes the total weighted earliness and tardiness cost over all jobs. Each unit of earliness is penalized by weight α_i, and each unit of tardiness by weight β_i.

$$\min f(Z) = \sum_{i=1}^{n} (\alpha_i E_i + \beta_i T_i) \tag{3}$$

Depending on their magnitudes, the ET penalties may be symmetric ($\alpha_i = \beta_i$) or asymmetric ($\alpha_i \neq \beta_i$) (Gordon *et al.*, 2002). In this work, we consider the asymmetric case, where the two penalties differ to reflect distinct priorities, depending on the application context. For instance, early completion may be less critical in a planned maintenance scheduling scenario than delays, as tardiness could lead to unexpected system failures or costly downtime.

To illustrate how job ordering affects earliness and tardiness, consider a single machine processing two jobs, J_1 and J_2. Job J_1 has processing time $p_{11} = 3$, release date $r_1 = 1$, due date $d_1 = 5$, and penalties $\alpha_1 = 2$ and $\beta_1 = 1$. Job J_2 has processing time $p_{21} = 2$, release date $r_2 = 0$, the same due date $d_2 = 5$, and penalties $\alpha_2 = 2$ and $\beta_2 = 1$. Since J_2 is available earlier, it may start at time zero, whereas J_1 cannot begin before time one.

Consider first the sequence $[J_2, J_1]$. Job J_2 completes at time $C_2 = 2$, yielding $E_2 = 3$ and $T_2 = 0$. Job J_1 begins at time 2 and completes at $C_1 = 5$, resulting in $E_1 = 0$ and $T_1 = 0$. For the alternative sequence $[J_1, J_2]$, job J_1 starts at its release date and completes at $C_1 = 4$, giving $E_1 = 1$ and $T_1 = 0$. Job J_2 then starts at time 4 and completes at $C_2 = 6$, producing $E_2 = 0$ and $T_2 = 1$. This simple example demonstrates how different sequencing decisions, while still respecting release dates, yield different earliness and tardiness values, highlighting the importance of ordering decisions in ET scheduling environments.

5. LINEAR PROGRAMMING MODEL FOR START TIME COMPUTATION AND OBJECTIVE FUNCTION EVALUATION

In this section, we present the LP model embedded within the metaheuristic and constructive algorithms to compute job start times on machines and to evaluate the TWET, which is used as the fitness function. Given a fixed job sequence generated by the metaheuristics, the LP model determines the corresponding optimal timing decisions, ensuring feasibility with respect to release dates and machine availability. In this way, the LP separates sequencing decisions from timing optimization, allowing each candidate solution to be evaluated accurately through a single scalar fitness value, while also providing complete scheduling information. Solving the model as a standalone model for the whole problem would be computationally intractable due to the combinatorial nature of sequencing and assignment decisions. However, when embedded within the metaheuristic framework, the LP remains feasible and convex for fixed job sequences, enabling exact computation of start times and objective function values.

The model's parameters are defined as follows. Let M denote the set of m unrelated parallel machines and let M_k represent the ordered sequence of jobs assigned to machine k, where $k \in M$. Each job j occupying a position in the sequence M_k corresponds to a job $J_i \in J$. The processing time of job j on machine k is denoted by p_{jk}. The parameters r_j and d_j represent the release date and due date of job j, respectively. The weights α_j and β_j indicate the earliness and tardiness penalties associated with job j.

The LP model includes the following decision variables. The continuous variable S_{jk} denotes the start time of job j on machine k, while C_{jk} denotes the completion time. The non-negative variables E_{jk} and T_{jk} represent, respectively, the earliness and tardiness of job j when processed on the machine k, measured in time units.

The LP model is formulated as follows:

$$\min \quad \sum_{k=1}^{m} \sum_{j=1,\lfloor M_k \rfloor \neq 0}^{\lfloor M_k \rfloor} \left(\alpha_j E_{jk} + \beta_j T_{jk}\right) \tag{4}$$

$$s.t. \quad C_{jk} = S_{jk} + p_{jk}, \;\; k \in M, \;\; j \in M_k, \;\; \lfloor M_k \rfloor \neq 0 \tag{5}$$

$$S_{jk} \geq r_j, \;\; k \in M, \;\; j \in M_k, \;\; \lfloor M_k \rfloor \neq 0 \tag{6}$$

$$S_{jk} \geq C_{(j-1)k}, \;\; k \in M, \;\; j \in M_k \setminus \{1\}, \;\; \lfloor M_k \rfloor \neq 0 \tag{7}$$

$$C_{jk} + E_{jk} - T_{jk} = d_j, \;\; k \in M, \;\; j \in M_k \tag{8}$$

$$C_{jk}, S_{jk}, E_{jk}, T_{jk} \geq 0, \;\; k \in M, \;\; j \in M_k, \;\; \lfloor M_k \rfloor \neq 0 \tag{9}$$

The objective function in Equation (4) minimizes the total weighted earliness and tardiness over all jobs. Constraint (5) establishes the completion time of each job as the sum of its start time and processing duration. Constraint (6) enforces the release date requirement, ensuring that no job starts before the release date. Constraint (7) imposes sequencing logic on each machine by ensuring that a job cannot start until its immediate predecessor has completed in the assigned sequence, thereby preventing job overlap. Constraint (8) links completion times to the corresponding earliness and tardiness variables, ensuring a consistent representation of timing deviations with respect to due dates. Finally, Constraint (9) enforces non-negativity on all decision variables.

6. METAHEURISTICS FOR UNRELATED PARALLEL MACHINES SCHEDULING PROBLEM

In this section, we present the metaheuristic algorithms employed in this study: GA, TS, and VNS. All three algorithms use the same structure to represent individuals (i.e., candidate solutions). First, we formalize the solution encoding and describe the decoding process. Each metaheuristic is then explained in detail, along with its associated procedures. Finally, we specify the stopping criterion adopted for all algorithms.

6.1. Encoding and Decoding Solution

We adopt a widely used representation for scheduling problems on parallel machines, where a vector of dimension m corresponds to the available machines. Each element of this vector represents a machine, and for each machine, a subordinate vector (possibly empty) specifies the sequence of jobs assigned to it.

Formally, an individual is encoded as an assignment vector $Z = [Z_1, Z_2, \dots, Z_m]$, where Z_k denotes the ordered sequence of jobs allocated to machine k, defined as

$$Z_k = \left[W_{1k}, W_{2k}, \dots, W_{p_k k}\right] \text{ and } W_{jk} = J_i, \text{ where } J_i \in J, \forall j \; \forall k \tag{10}$$

Here, p_k represents the number of jobs assigned to machine k, and $W_{jk} = J_i$ indicates that job J_i is scheduled in the j-th position on machine k. This genotypic representation fully specifies the processing sequence on each machine.

To ensure feasibility, the complete set of job J must be assigned to the available machines without duplication. This condition is enforced by requiring that

$$\bigcup_{k=1}^{m} Z_k = J \text{ and } Z_k \cap Z_{k'} = \emptyset, \forall k \neq k' \tag{11}$$

To obtain a complete phenotypic solution, the decoding process transforms the genotype Z into a feasible schedule by solving the LP model presented in Section 5. This model computes the start times S_{jk} of jobs W_{jk} on machine k, so as to

minimize the TWET. The resulting timing information, combined with the assignment vector Z, it fully characterizes the final job schedule.

To demonstrate the encoding and decoding process, we consider an instance of the UPMSP comprising four jobs and two machines, as detailed in Table 2. For example, the table indicates that job J_3 has a release date $r_3 = 6$, and a due date $d_3 = 8$. The associated earliness and tardiness penalties are $\alpha_3 = 2$ and $\beta_3 = 3$, respectively. The processing times for this job are $p_{31} = 3$, on machine 1 and $p_{32} = 2$ on machine 2.

Table 2. UPMSP input instance (4 jobs, 2 machines) used for demonstrating encoding and decoding steps.

Job	**Processing time**		**Release date**	**Due date**	**Weights**	
J_i	M_1	M_2	r_i	d_i	α_i	β_i
J_1	2	3	3	5	1	1
J_2	3	3	3	6	2	1
J_3	3	2	6	8	2	3
J_4	3	2	6	8	3	2

An example of solution encoding is given by $Z = [[1,4],[2,3]]$. The set of jobs $J = \{1,2,3,4\}$ is distributed across two machines and structured as a vector $Z = [Z_1, Z_2]$. In this case, $Z_1 = [W_{11}, W_{21}]$ and $Z_2 = [W_{12}, W_{22}]$, where Z_1 and Z_2 denotes the sequence of jobs assigned to machines 1 and 2, respectively. Specifically, $Z_1 = [W_{11}, W_{21}] = [1,4]$ and $Z_2 = [W_{12}, W_{22}] = [2,3]$. Each element W_{jk} represents job J_i assigned to machine k at position j within Z_k. Thus, machine 1 processes job 1 followed by job 4, while machine 2 processes job 2 followed by job 3.

Given the input data and the encoded vector Z, the LP model is applied, producing the vector Z_{seq}, which specifies the start and completion times of each job:

$$Z_{seq} = [[(3,5),(6,9)],[(3,6),(6,8)]].$$

Together, (Z, Z_{seq}) fully specifies the decoded scheduling, yielding an objective value of $f(Z) = 2$. The corresponding Gantt chart is shown in Figure 1.

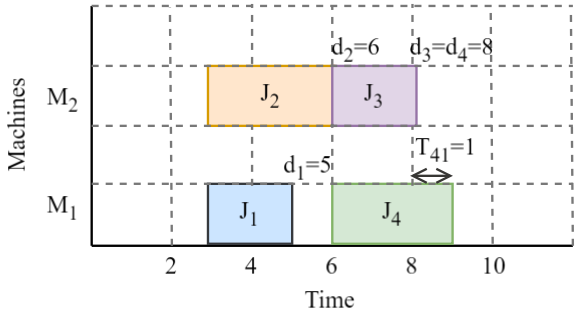

Figure 1. Decoded schedule for the illustrative instance in Table 2 obtained from $Z = [[1,4],[2,3]]$. Job 4 is late by one unit of time, resulting in a TWET value of $f(Z) = 2$.

6.2. Genetic Algorithm

The GA starts with a randomly generated population. Each individual is decoded and evaluated using the embedded LP model to ensure feasibility and compute an exact objective value. At each generation, a subset of the best individuals is preserved, and the remaining population is generated via elitism, tournament selection, crossover, and mutation. The offspring are then re-evaluated by the LP model, and the process is repeated until the stopping criterion is satisfied. The pseudo-code is provided in Algorithm 1, and further details are presented in the subsequent subsections.

Input: Problem data
Output: Best solution Z_{best} and its objective value $f(Z_{best})$

1: Generate an initial population with $PopSize$ random candidate solutions
2: Evaluate each candidate using the LP model
3: Set $Z_{best} \leftarrow$ best solution in the current population
4: Set $f(Z_{best}) \leftarrow$ objective value of Z_{best}
5: **while** the stopping criterion is not satisfied, **do**
6: Select R individuals for the next generation using an elitist strategy
7: Select parents using selection mechanisms (elitism and tournament)
8: Perform crossover on pairs of parents with crossover rate P_c
9: Apply mutation to the resulting offspring with mutation rate P_m
10: Evaluate new candidates using the LP model
11: Set $Z^* \leftarrow$ best solution in the new population
12: Set $f(Z^*) \leftarrow$ objective value of Z^*
13: **if** $f(Z^*) < f(Z_{best})$ **then**
14: Set $Z_{best} \leftarrow Z^*$
15: Set $f(Z_{best}) \leftarrow f(Z^*)$
16: **end if**
17: **end while**
18**: return** Z_{best}, $f(Z_{best})$

Algorithm 1. Genetic Algorithm with LP-based decoding

6.2.1. Selection

The GA's selection process determines which individuals from the current population will be chosen as parents to generate the next generation. The proposed GA employs a hybrid strategy that combines elitism and tournament selection.

Elitism directly preserves the top-performing individuals. Specifically, the best R individuals, corresponding to a proportion of the population size (i.e., $r_e \times PopSize$), are retained unchanged in the subsequent generation. This mechanism guarantees that the most fit solutions persist, promoting convergence toward optimal regions of the solution space.

The remaining individuals in the population are generated through reproduction, using a probabilistic combination of parent selection mechanisms. With probability P_e, parents are randomly selected from the elite pool. With complementary probability $1 - P_e$, parents are selected using tournament selection.

In the tournament selection procedure, a group of t individuals (with $t = 5$ in this study, as suggested by Eiben and Smith (2015), to increase selection pressure) is randomly sampled from the population. The individual with the best fitness value within this group is selected as a parent.
This hybrid strategy preserves the genetic material of high-quality individuals while encouraging diversity and exploration of new areas in the solution space.

6.2.2. Crossover

We use a machine-based crossover called Machine-Based Job Subset Crossover (MBJSC) that is tailored to a machine-oriented chromosome representation. With probability P_c, one machine is randomly selected from each parent (re-sampling if both are empty). A contiguous job sequence is then extracted from each selected machine and exchanged to form two offspring. If only one of the selected machines is non-empty, its sequence is transferred to the empty machine. Figure 2 illustrates the chromosome structure before and after the MBJSC process, and Algorithm 2 summarises this procedure.

A repair step is then applied to remove duplicates and reinsert missing jobs, ensuring that each job appears only once.

Input: Two parents' schedules: Parent1, Parent2
Output: Two offspring schedules: Child1, Child2

1: Initialise Child1 ← Parent1, Child2 ← Parent2
2: Select a machine randomly from Parent1
3: Select a machine randomly from Parent2

4: **while** both selected machines are empty, **do**
5: Re-select the two machines
6: **end while**
7: **if** only one selected machine contains jobs, **then**
8: Extract a job subset from the non-empty machine
9: Transfer the subset to the corresponding machine in the offspring
10: Remove the subset from the original machine
11: **else**
12: Extract job subsets from each selected machine
13: Exchange the two subsets between the corresponding offspring machines
14: **end if**
15: Apply the repair procedure to Child1 and Child2
16: **return** Child1, Child2

Algorithm 2. Machine-Based Job Subset Crossover (MBJSC) operator

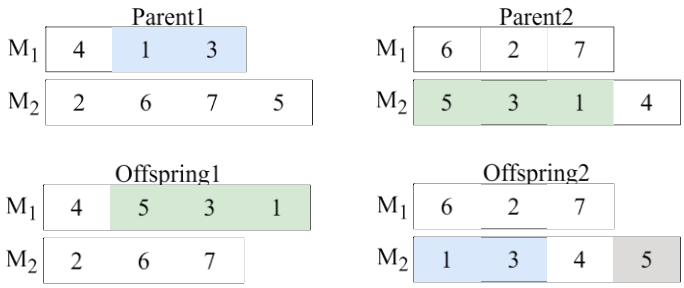

Figure 2. Illustration of Machine-Based Job Subset Crossover (MBJSC). A contiguous job sequence is selected from one machine in each parent and exchanged to form two offspring (green and blue sequences). The subsequent repair step removes duplicate jobs and reinserts the missing job (shown in grey).

6.2.3. Mutation

Two mutation operators are used with probability P_m: Internal Rearrangement (IR), and Redistribution Across Machines (RDAM). Figure 3 depicts the chromosome structure before and after each mutation.

- IR: If the chosen machine contains at least two jobs, two of them are selected at random and swapped. This promotes the exploration of alternative job sequences within a single machine.
- RDAM: A job is randomly selected from one machine and reassigned to another. This facilitates the discovery of new inter-machine allocation configurations.

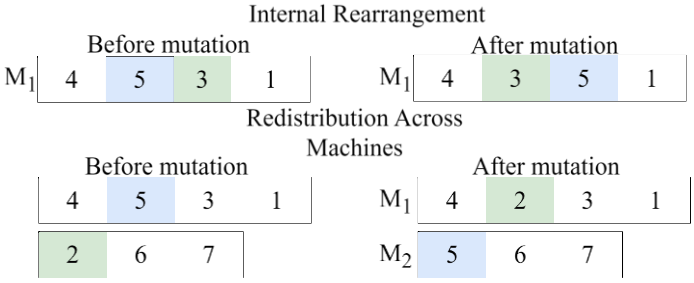

Figure 3. Illustration of the mutation operator used in GA: IR (top) swaps two jobs (showing in green and blue) within the same machine, and RDAM (bottom) exchanges jobs between two machines.

6.3. Tabu Search Algorithm

The TS algorithm begins with a randomly generated initial solution, which is then iteratively explored to identify improved alternatives. A central component of the method is the tabu list (TL). This records recently visited solutions represented through their complete machine–job assignments, to prevent cycling and promote exploration beyond local optima. A neighbor solution is classified as tabu if its encoded representation appears in the TL, unless it satisfies an aspiration criterion by improving upon the best solution found so far.

The quality of each candidate solution is evaluated using an LP model to ensure consistency with the problem objective. The search process is structured around two main mechanisms:

- Exploration: At each iteration, a set of neighboring solutions is generated using multiple operators that perturb the current solution, promoting a broad exploration of the solution space.
- Diversification: If the algorithm fails to improve after a predefined number of iterations, indicating stagnation, a diversification strategy is triggered. This strategy involves generating a new random solution to redirect the search towards unexplored regions.

The algorithm continues until the stopping criterion is satisfied. The complete procedure is detailed in Algorithm 3.

Input: Problem data
Output: Best solution Z_{best} and its objective value $f(Z_{best})$

1: Generate an initial solution Z randomly
2: Evaluate $f(Z)$ using the LP model
3: Set $Z_{best} \leftarrow Z$
4: Set $f(Z_{best}) \leftarrow f(Z)$
5: **while** the stopping criterion is not satisfied, **do**
6: Remove expired movements from the tabu list (TL).
7: Generate a set of neighboring solutions using distinct operators.
8: Evaluate each neighbor's objective value using the LP model.
9: Select the best neighbor solution Z^* such that:
10: $Z^* \notin TL$ **or** $\left(Z^* \in TL$ **and** $f(Z^*) < f(Z_{best})\right)$
11: **if** $f(Z^*) < f(Z_{best})$ **then**
12: Set $Z_{best} \leftarrow Z^*$
13: Set $f(Z_{best}) \leftarrow f(Z^*)$
14: **end if**
15: Add Z^* to the TL
16: **If** diversification conditions are met, **then**
17: Generate a new random candidate solution Z
18: **end if**
19: **end while**
20: **return** Z_{best}, $f(Z_{best})$

Algorithm 3. Tabu Search with LP-based evaluation of neighbor solutions

6.3.1. Neighborhood Generation

Neighborhood generation determines how the search space is explored. At each iteration, a set of candidate neighbors is generated by applying a perturbation operator to the current solution. The proposed method uses six neighborhood operators that combine traditional mechanisms with custom strategies to enhance diversification. The standard operators are:

- Intra-machine swap: Two jobs are randomly selected within the same machine, and their positions are swapped.
- Inter-machine swap: A job from one machine is exchanged with a job from another machine.
- Insert job: A job is removed from one machine and inserted into an arbitrary position in another machine's job sequence.
- Move job: A job is selected from one machine and appended to the end of another machine's queue.

In addition to these, we propose two custom operators:

- Balance load: This operator addresses workload imbalance. The machines with the highest and lowest job counts are identified. A job is removed from the most loaded machine and reassigned to the least loaded one, encouraging the search towards more balanced schedules.
- Chain swap: Three machines are selected at random. The first job in each of their queues (if available) is temporarily removed. These jobs are then reassigned in a cyclic manner; each being placed at the end of the next machine in the sequence. This operator introduces more complex, system-wide reallocations unlikely to arise from pairwise exchanges.

To generate neighbors, an operator is selected uniformly at random and applied to the current solution. This process is repeated until the required number of admissible neighbors has been obtained. Neighbors that have already been considered are ignored.

6.4. Variable Neighborhood Search algorithm

The VNS algorithm follows the classical framework established in the literature, alternating between systematic changes in neighborhood structures and local search phases. The search is initialised with a solution constructed using the Minimum Weighted Slack (MWS) dispatching rule. Notably, the VNS benefits from the same neighborhood set (including the proposed balance load and chain swap operators) and from the LP-based exact evaluation used throughout the search.

6.4.1. Minimum Weighted Slack

The initial solution is constructed using the MWS rule, a well-established dispatching rule in scheduling theory (Pinedo, 2022). For each job j on machine m, the weighted slack is computed according to Equation (12).

$$Slack_{jm} = \frac{d_j - r_j - p_{jm}}{\alpha_j + \beta_j} \tag{12}$$

where d_j is the due date, r_j is the release time, p_{jm} is the processing time of job j on machine m, and α_j and β_j are the earliness and tardiness weights, respectively.

Each job is iteratively assigned to the machine that minimizes the weighted slack. To promote workload balance, a priority queue is maintained to track the cumulative load of each machine, ensuring that jobs are preferentially allocated to the least-loaded machine.

6.4.2. Variable Neighborhood Search algorithm

The VNS metaheuristic begins with an initial solution constructed using the MWS dispatching rule. It then iteratively explores predefined neighborhood structures, described in Section 6.3.1, to improve the solution quality. Each candidate solution is evaluated using the embedded LP model. The search process terminates when the stopping criterion is satisfied. The overall procedure is outlined in Algorithm 4.

The set of neighborhoods $\mathcal{N}$, represented in Equation (13), consists of several operators that perturb the current solution by altering job assignments and sequences across machines. Each neighborhood $\mathcal{N}_k \in \mathcal{N}$ defines a specific transformation applied to the incumbent solution Z to generate a candidate solution Z^*. These neighborhoods are explored sequentially, and each candidate is refined through a local search phase before its objective value is evaluated.

N = {swap-intra machine, swap inter-machine, insert job, move job, balance load, chain swap} (13)

Input: Problem data
Output: Best solution Z_{best} and its objective value $f(Z_{best})$

1: Generate an initial solution Z using the MWS heuristic
2: Evaluate $f(Z)$ using the LP model
3: Set $Z_{best} \leftarrow Z$
4: Set $f(Z_{best}) \leftarrow f(Z)$
5: **while** the stopping criteria are not satisfied, **do**
6: **For each** neighborhood structure $\mathcal{N}_k \in \mathcal{N}$ **do**

7: Generate neighbor solution $Z^* \leftarrow \mathcal{N}_k(Z_{best})$
8: Apply local search to refine Z^*
10: Evaluate $f(Z^*)$ using the LP model
11: **if** $f(Z^*) < f(Z_{best})$ **then**
12: Set $Z_{best} \leftarrow Z^*$
13: Set $f(Z_{best}) \leftarrow f(Z^*)$
14: **end if**
15: **end while**
16: **return** Z_{best}, $f(Z_{best})$

Algorithm 4. Variable Neighborhood Search with LP-based evaluation

6.4.3. Local Search Procedure

The local search procedure, detailed in Algorithm 5, refines a candidate solution by exploring its immediate neighborhood. At each iteration, several neighboring solutions (up to a predefined maximum) are generated through small perturbations by the neighborhood operators. The best neighbor, according to the objective value, is selected and retained. The process continues until no further improvement is observed.

Input: Incumbent solution Z, objective value $f(Z)$, maximum number of neighbors max $\_neighbours$
Output: Refined solution Z_{best} and its objective value $f(Z_{best})$

1: Set $Z_{best} \leftarrow Z$ and
2: Set $f(Z_{best}) \leftarrow f(Z)$
3: **for** i=1 **to** *max_neighbors* **do**
4: Generate a neighbor solution $Z^* \leftarrow \mathcal{N}_k(Z_{best})$
5: Evaluate $f(Z^*)$ using the LP model
6: **if** $f(Z^*) < f(Z_{best})$ **do**
7: Set $Z_{best} \leftarrow Z^*$
8: Set $f(Z_{best}) \leftarrow f(Z^*)$
9: **end if**
10: **end for**
11: **return** Z_{best}, $f(Z_{best})$

Algorithm 5. Local Search Procedure used in VNS

6.5. Stopping Criteria

A common stopping criterion has been adopted to ensure a fair and consistent comparison between all metaheuristic algorithms. The search process terminates when one of the following two conditions is met:

(i) The maximum number of iterations (maxGen) is reached, or
(ii) The algorithm performs a predefined number of consecutive iterations without improving upon the best solution (maxStagnantIter).

These criteria aim to balance the exploration with computational efficiency to prevent excessive runtimes.

7. NUMERICAL EXPERIMENTS

7.1. Software Implementation Details

All metaheuristics were implemented in Python 3.9.6 using Visual Studio Code as the development environment and executed on a workstation equipped with an Intel® Core™ i7-8750H CPU @ 2.20 GHz and 12 GB RAM. Python was selected for its extensive scientific computing ecosystem and suitability for rapid prototyping of heuristic algorithms. The linear programming (LP) model was implemented using the PuLP library and solved with IBM ILOG CPLEX Optimization Studio 22.1.1, which was chosen for its reliability and efficiency in solving medium- to large-scale linear optimization problems.

7.2. Test Instances

Currently, no publicly available benchmark instances encompass all the integrated features of the problem addressed in this work. In order to compare the metaheuristics, we have developed a set of test instances that reflect the complexity of the scheduling environment considered. These instances were generated based on the structural guidelines and data ranges proposed by Bank and Werner (2001) and Kim *et al.* (2012), which are recognized as reference works in the UPMSP literature. The adoption of these guidelines ensures consistency in benchmark generation while allowing sufficient flexibility to accommodate the additional problem characteristics.

Unlike those previous studies, the proposed instances explicitly incorporate all key problem features, with each instance defining a unique combination of jobs characterized by job-dependent release dates, due dates, machine-dependent processing times, and cost coefficients for earliness and tardiness. Processing times were initialized by drawing base values from a uniform distribution in the interval $[1,50]$, a range that models heterogeneous job sizes while remaining compatible with commonly adopted processing-time scales in the literature. To model machine-dependent variability, each job's processing time on a given machine was perturbed using a multiplicative factor randomly selected from the interval $[0.9,1.1]$. This limited perturbation reflects differences in performance among the machines without introducing excessive imbalance.

Due dates were generated using a modified version of the expression proposed by Bank and Werner (2001) for problems with common due dates. The reference due date was defined as $d = \left\lfloor \frac{2}{3}\frac{n}{m}\bar{p} \right\rfloor$, where $\bar{p} = \frac{\sum_{i=1}^{n}\sum_{k=1}^{m} p_{ik}}{nm}$ is the average processing time across all jobs and machines. This formulation yields due dates that are neither overly restrictive nor excessively loose. Each job's actual due date, d_i, is then randomly adjusted within a $\pm 10\%$ range of the reference value. Release times were also randomly generated for each job within the interval $[0, d_i - p_{min}]$, where p_{min} is the minimum processing time of that job across all machines. This guarantees that the job can be completed by its due date on at least one machine. Lastly, the cost weights for earliness and tardiness were assigned using a uniform distribution in the interval $[1,3]$, preventing extreme penalty dominance and remaining consistent with previous computational studies.

7.3. Parameter Settings

The parameter settings for each metaheuristic algorithm were determined based on a combination of values reported in the literature (Eiben and Smith, 2015; Li and Gao, 2016; Fan *et al.*, 2023) and empirical exploratory tuning through controlled experimentation. The purpose of this tuning phase was to select suitable parameter configurations for the experiments. All algorithms were configured to perform approximately the same number of objective function evaluations to ensure a fair comparison.

A preliminary tuning phase was conducted using smaller problem instances, specifically those with job sizes $n \in \{40, 60\}$, and machine counts $m \in \{6,8\}$. For each algorithm, a set of parameter combinations was evaluated. Each configuration was executed five times, and the one achieving the best average objective function value across these runs was selected.

For the GA, the explored combinations included population sizes $\text{PopSize} \in \{200,400,600\}$, crossover probabilities $P_c \in \{0.8,0.9,0.95\}$, and mutation probabilities $P_m \in \{0.2,0.3,0.4\}$. The selected configuration used a population size of $\text{PopSize}=400$ individuals, a crossover probability $P_c = 0.9$, and a mutation probability $P_m = 0.2$. An elitism rate $r_e = 0.1$ was applied, meaning that 10% of the best individuals were preserved in each generation. Parent selection followed a hybrid strategy: with a probability $P_e = 0.3$, tournament selection with a pool size $t = 5$; otherwise, elitism was employed. The MBJSC crossover operator was adopted, and the mutation was determined probabilistically; either the IR or RDAM mutation was selected with equal probability.

For TS, the tuning process considered tabu list sizes $|TL| \in \{200,300,400\}$. A tabu list size of 400 produced the best average results and was therefore adopted. The tabu duration was fixed at 60 iterations, and a diversification mechanism was introduced every 30 iterations.

For VNS, the number of neighbors explored in each local search phase was varied among $\text{max\_neighbors} \in \{50,70,90\}$. The final configuration adopted $\text{max\_neighbors} = 70$, with a perturbation strength of 30 to control the shaking process.

All metaheuristics employed the same stopping criteria. The stopping criterion combined a maximum number of iterations, set at $\text{maxGen} = 200$, and a stagnation threshold of $\text{maxStagnatIter} = 40$ consecutive iterations without improvement in the objective function value.

7.4. Computational Results and Analysis

To evaluate the effectiveness of the metaheuristic algorithms, 20 test instances of varying dimensions were considered, as previously described. Each instance was solved five times using each metaheuristic to account for variability in the results. The performance assessment was based on the objective function value:

$$f(Z) = \sum_{i=1}^{n} (\alpha_i E_i + \beta_i T_i) \tag{14}$$

To assess the quality of the obtained solutions relative to the best-known value, the Average Percentage Deviation (APD) was computed for each instance. The best-known value, f_{BKV}, was defined as the best objective value found across all runs and all metaheuristics. The APD was calculated as follows:

$$APD = \frac{1}{N_R} \sum_{l=1}^{N_R} \frac{f_l - f_{BKV}}{f_{BKV}} \times 100\% \tag{15}$$

where $N_R = 5$ denotes the number of runs per instance, f_l is the objective value of the best solution obtained in the l-th run, and f_{BKV} is the best-known objective value across all runs and metaheuristics.

Table 3 presents the computational results obtained by the three metaheuristics, reporting the average objective values, APD, and computational time across all tested instances. The TS algorithm consistently achieves the lowest average objective function values f_{avg}, indicating its superior performance in finding high-quality solutions.

In terms of APD, TS demonstrates the lowest deviations from the best-known solutions. For instance, the APD drops to 0.7% for the instance with $n = 100$, $m = 6$, and to 1.2% for $n = 120$, $m = 8$. In contrast, GA and VNS exhibit larger deviations. The GA, in particular, reaches a maximum APD of 20.2% for the instance with $n = 120$, $m = 6$. Although VNS generally yields higher APD values, its average objective function values are marginally better than those of GA in larger problem instances.

Overall, these results indicate that TS offers a more robust and consistent performance across all tested configurations, particularly as the problem size increases.

Regarding computational time, the results indicate that execution time increases with problem size. The influence of the number of machines is less pronounced and not strictly monotonic, reflecting the varying convergence behaviors observed across different instances.

Table 3. Computational results comparing GA, TS, and VNS across different problem instances. The table reports the average objective function value (f_{avg}), the Average Percentage Deviation (APD, %) relative to the best-known value, and the average computational time (in seconds) obtained over five independent runs for each instance.

		f_{avg}			APD(%)			Time (s)		
n	m	GA	TS	VNS	GA	TS	VNS	GA	TS	VNS
40	6	2015.0	2007.6	2037.0	5.3	4.9	6.5	3490.6	2239.0	2686.8
	8	1049.4	1062.6	1092.8	2.3	3.6	6.5	3567.1	3317.0	2891.3
	10	750.4	731.2	772.4	5.1	2.4	8.2	3981.2	3148.9	2987.5
	12	663.0	667.6	697.4	3.1	3.8	8.5	3429.5	2884.8	2891.6
60	6	3090.2	2918.4	3096.0	7.3	1.3	7.5	3845.6	4449.6	3391.1
	8	2697.4	2602.0	2797.4	7.1	3.3	11.1	4180.1	4049.1	3551.7
	10	2034.0	1944.2	2103.8	6.3	1.6	9.9	4528.5	4310.7	3416.7
	12	2656.0	2608.0	2764.6	6.0	4.1	10.3	4833.6	4438.8	3247.7
80	6	6931.8	6282.4	6822.0	12.0	1.5	10.2	4974.1	5966.7	3153.9
	8	5545.2	5194.2	5553.8	9.0	2.1	9.2	5197.8	5122.7	3811.5
	10	4074.6	3643.6	3961.2	16.3	4.0	13.0	4424.3	6191.8	3398.2
	12	2679.4	2568.6	2706.0	6.6	2.2	7.7	4626.8	5411.9	3432.1
100	6	10530.4	9409.0	10115.2	12.7	0.7	8.2	5062.7	4611.0	3138.3
	8	8249.2	7381.4	8070.8	13.7	1.7	11.2	4874.3	4387.6	3133.5
	10	5897.8	5310.0	5637.8	12.7	1.5	7.8	4519.3	4269.3	3521.3

		f_{avg}			APD(%)			Time (s)		
n	m	GA	TS	VNS	GA	TS	VNS	GA	TS	VNS
120	12	5645.6	5019.6	5497.6	14.2	1.5	11.2	4802.5	4337.4	3559.9
	6	14861.8	12582.0	13702.6	20.2	1.7	10.8	5047.6	4407.9	3094.9
	8	11793.2	10060.4	10993.0	18.7	1.2	10.6	4837.6	4650.8	3313.1
	10	8689.8	7643.0	8256.4	14.7	0.9	9.0	4814.9	4593.0	3397.2
	12	8227.2	7183.4	7922.2	17.4	2.5	13.0	4818.9	4720.3	3305.1

To assess the performance variability of the metaheuristics, the APD was analyzed. Figures 4 and 5 present the mean APD values together with their corresponding 95% confidence intervals (CI) across different problem sizes.

Figure 4 illustrates the results for problem sizes $n = 40$ and $n = 60$. TS consistently achieves the lowest APD values, with particularly robust performance for $n = 60$, where its APD remains close to zero across several configurations. GA reveals moderate APD values, relatively narrow confidence intervals for $n = 40$. In contrast, VNS generally records the highest APD values, especially as the number of machines increases.

Figure 5 extends the analysis to larger problem sizes ($n = 80$, $n = 100$, and $n = 120$), confirming the trends observed for smaller instances. As problem size increases, both GA and VNS tend to produce higher APD values, with GA showing the largest deviations overall. TS, by contrast, maintains comparatively low APD values with stable confidence intervals, demonstrating robustness and scalability. Notably, for $n = 100$ and $n = 120$, the confidence intervals for VNS become narrower, and its APD values are moderated relative to those of GA, although they remain higher than those of TS.

Overall, the confidence interval analysis highlights the consistent superiority of the TS algorithm in terms of solution quality and reliability across all problem sizes.

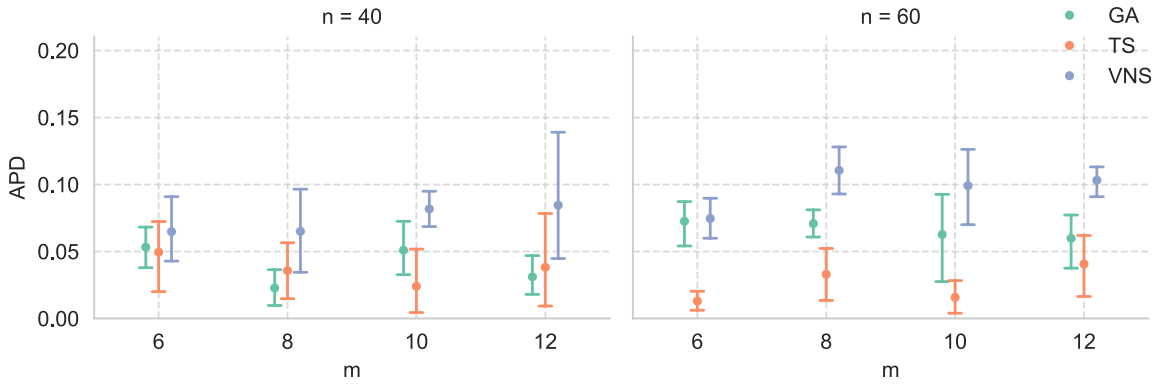

Figure 4. Interval plot of APD values for GA, TS, and VNS on instances with n=40 and n=60, with 95% CI for the mean.

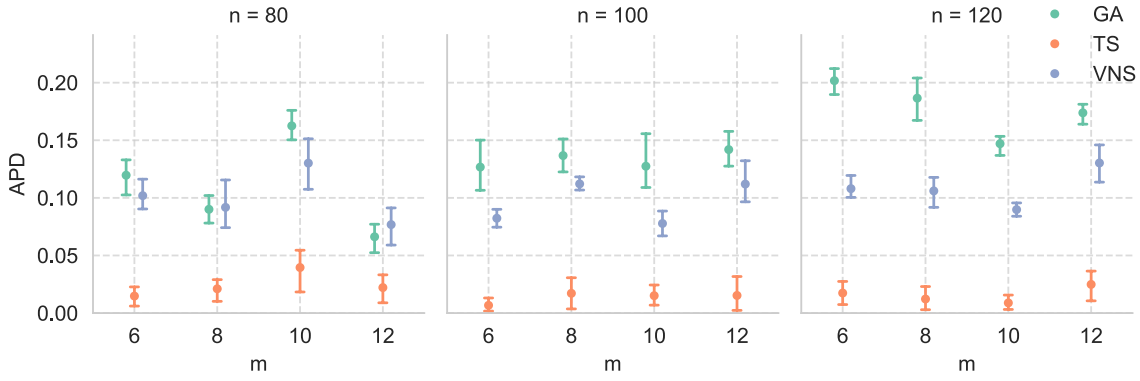

Figure 5. Interval plot of APD values for GA, TS, and VNS on instances with n=80, n=100, and n=120, with 95% CI for the mean.

To compare the three metaheuristics, $N = 20$ distinct instances were considered. The median objective function value across five independent runs was used as the representative performance measure for each instance. All statistical tests were performed using a significance level of $\alpha = 0.05$.

A Friedman test was first conducted to determine if there were significant differences in performance across the three metaheuristics. The test revealed a significant difference ($\chi^2 = 24.4, p < 0.001,\ W = 0.610$), indicating that the choice of algorithm has an impact on the solution quality. Kendall's $W = 0.610$ indicates a strong effect size, confirming a high degree of consistency in the algorithms' rankings across all considered instances.

To identify which specific metaheuristics differed, a post-hoc analysis was performed using Wilcoxon signed-rank tests with Holm-Bonferroni correction to adjust for multiple comparisons. The results show that the TS significantly outperformed both VNS ($\mathrm{Z} = -4.427, \mathrm{p} < 0.001, \mathrm{r} = 0.70$) and GA ($\mathrm{Z} = 4.111, \mathrm{p} < 0.001, \mathrm{r} = 0.65$). In both cases, the large effect sizes (r) indicate a substantial and consistent performance advantage for TS.

Conversely, the comparison between VNS and GA yielded no statistically significant difference ($\mathrm{Z} = -0.316, \mathrm{p} = 0.752, \mathrm{r} = 0.05$), with a negligible effect size. This suggests that while TS is the superior choice for the problems tested, GA and VNS do not exhibit statistically significant differences in performance.

To complement the quantitative analysis, a Gantt chart showing the optimal solution is provided. Figure 6 illustrates the schedule obtained for the instance with $n = 60$ jobs and $m = 8$ machines, corresponding to the best solution identified across all metaheuristics. In this instance, the best-performing solution was produced by the TS algorithm.

The Gantt chart offers a visual interpretation of the final schedule, highlighting job assignments and machine utilization over time.

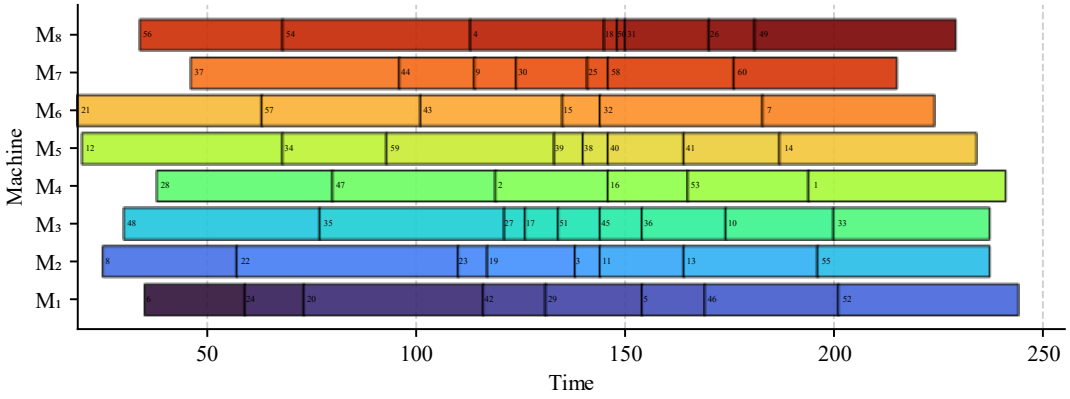

Figure 6. Gantt chart of the best schedule obtained for the instance with n = 60 jobs and m = 8 machines, produced by the TS algorithm.

7.5. Convergence Analysis

To evaluate the convergence speed of the metaheuristics, we considered the average number of iterations (across five runs per test instance) needed to solve within an absolute deviation of no more than 25% of the best-known value. The corresponding standard deviation was also reported to assess variability in convergence behavior.

Figure 7 presents the average iteration counts and standard deviations for problem instances with $n = 40$ and $n = 60$. For $n = 40$, the TS consistently outperforms GA and VNS, achieving the target performance in fewer iterations on average. However, VNS exhibits significant variability, particularly for the instance $(40,8)$, indicating inconsistent convergence behavior. For $n = 60$, GA is the slowest to converge, requiring the highest number of iterations, whereas TS maintains its superior efficiency. VNS continues to express high standard deviations, underscoring the variability of its convergence speed on medium-sized problems.

Figure 8 presents the results for larger instances ($n = 80, 100, 120$). In these cases, GA shows a marked increase in the number of iterations required to reach the convergence threshold, with performance degrading as the problem size increases. TS remains the most efficient across all tested configurations, with both lower average iteration counts and smaller variability. VNS demonstrates improved performance relative to GA, especially in the largest instances, but still falls short of TS. Notably, GA presents high standard deviations in several cases (e.g.,$(120,8)$), suggesting divergence in convergence behavior across runs.

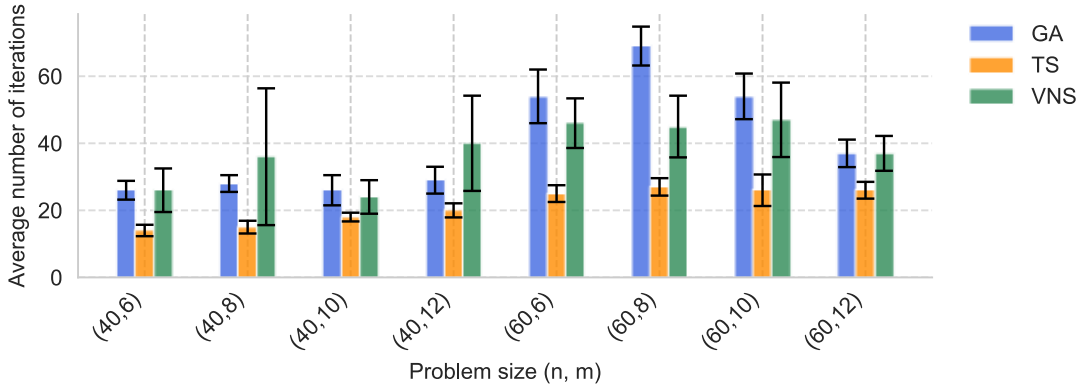

Figure 7. Average number of iterations required to reach a solution within 25% of the best-known objective value, together with the corresponding standard deviation, for GA, TS, and VNS on instances with n = 40 and n = 60.

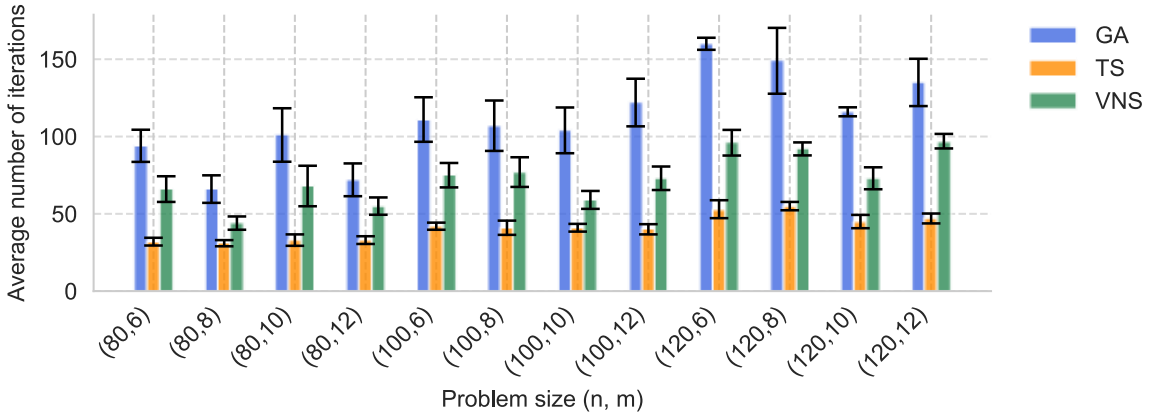

Figure 8. Average number of iterations required to reach a solution within 25% of the best-known objective value, together with the corresponding standard deviation, for GA, TS, and VNS on instances with n = 80, n = 100, and n = 120.

Across all problem sizes, TS proves to be the most efficient and stable metaheuristic, consistently requiring fewer iterations to reach the 25% deviation threshold. In contrast, GA shows slower convergence and high variability, particularly for larger instances. VNS exhibits intermediate convergence behavior, occasionally performing well, but with inconsistent results, especially in medium-sized instances where convergence behavior varies considerably.

These findings reinforce TS's superiority not only in terms of final solution quality, as previously shown, but also in terms of convergence efficiency and stability.

7.6. Convergence Analysis with Mixed-Integer Linear Programming Reference Solutions

To complement the analysis of convergence based on relative thresholds, an additional investigation was conducted on small problem instances with known reference solutions obtained from a MILP model. This analysis concentrates on the absolute convergence behavior of the metaheuristics concerning optimal or near-optimal solutions.

To this end, the model was implemented and solved using CPLEX to provide reference solutions for small instances. Due to the computational limitations of exact optimization, this analysis was restricted to instances with $n \leq 10$. Optimal solutions were obtained for instances with $n = 5$, while near-optimal solutions with residual gaps below 1% were considered for instances with $n = 10$. For completeness, the MILP formulation is provided in Appendix A.

Figure 9 shows the convergence behavior of the TS algorithm for two representative instances. The curves illustrate the median gap to the MILP reference solution over five independent runs as a function of the number of iterations.

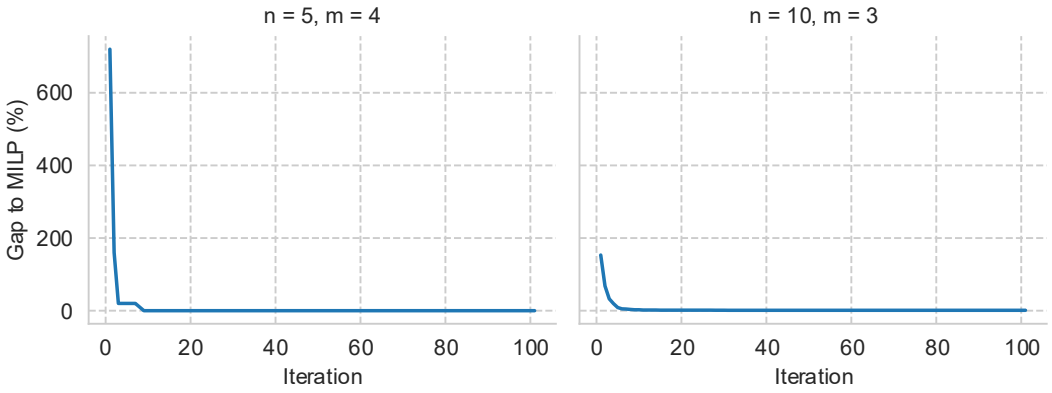

Figure 9. Convergence of the TS algorithm for small problem instances with MILP reference solutions: left, n = 5, m = 4; right, n = 10, m = 3. The curves show the median gap to the MILP reference value over five independent runs.

For the instance test with $n = 5$ and $m = 4$, TS reaches the optimal solution within a small number of iterations. For the instance test with $n = 10$ and $m = 3$, TS consistently converges to solutions within 1% of the MILP reference value, with most improvements occurring during the early stages of the search.

8. CONCLUSION

This study provides a comparative assessment of three metaheuristic approaches—Genetic Algorithm, Tabu Search, and Variable Neighborhood Search—applied to the unrelated parallel machine scheduling problem to minimize total weighted earliness and tardiness. The problem formulation integrates distinct due dates and non-zero release times, reflecting features observed in practical scheduling applications.

The novelty of this work resides in combining metaheuristic search with an embedded LP-based approach. This integration is used to determine job start times and to ensure an accurate evaluation of solution quality. In this way, metaheuristics can explore the solution space of a general UPMSP while maintaining precise computation of the objective function.

Computational results on medium- and large-scale instances demonstrate that Tabu Search consistently delivers superior performance in terms of both solution quality and convergence speed. In contrast, the genetic algorithm converges more slowly, a disadvantage that becomes more pronounced as the problem size increases. Variable Neighborhood Search achieves intermediate results but exhibits greater variability, particularly on medium-sized instances. These findings are supported by the statistical analysis, which confirms the robustness and reliability of Tabu Search across the tested instances.

Although the proposed methods target the general UPMSP, they are also applicable to more restrictive settings, such as identical parallel machines, zero release times, or common due dates. This versatility enhances the framework's applicability across a wide range of scheduling environments.

From an operational standpoint, the stable behavior of Tabu Search across different instance sizes suggests that it can be a viable option for decision support in practical scheduling applications.

A direction for future work is to evaluate the proposed framework on well-established public datasets for unrelated parallel machine scheduling problems, thereby reinforcing the external validity of the results and facilitating comparison with existing approaches.

Further extensions may focus on incorporating additional sources of realism, such as workforce-related constraints (e.g., operator availability or skill-dependent machine eligibility), or on replacing distinct due dates with due windows to represent delivery flexibility. In addition, multi-objective extensions that explicitly account for productivity–quality trade-offs constitute a promising and practically relevant direction for future work.

REFERENCES

Aalaei, A., Kayvanfar, V., and Davoudpour, H. (2017). A multi-objective optimization for preemptive identical parallel machines scheduling problem. Computational & Applied Mathematics, 36:1367-1387.

Alvarez-Valdes, R., Tamarit, J.M., and Villa, F. (2015). Minimizing weighted earliness-tardiness on parallel machines using hybrid metaheuristics. *Computers & Operations Research*, 54:1-11.

Amorim, R., Thomaz, M., and de Freitas, R. (2017). Solving large instances applying meta-heuristics for classical parallel machine scheduling problems under tardiness and earliness penalties. *2017 XLIII Latin American Computer Conference (CLEI)*, Cordoba, Argentina, 1-10.

Arik, O.A. (2020). Comparisons of metaheuristic algorithms for unrelated parallel machine weighted earliness/tardiness scheduling problems. *Evolutionary Intelligence*, 13(3):415-425.

Arık, O.A., Schutten, M., and Topan, E. (2022). Weighted earliness/tardiness parallel machine scheduling problem with a common due date. *Expert Systems with Applications*, 187: 115916

Asghari, M. and Nezhadali, S. (2014). Fuzzy Programming for Parallel Machines Scheduling: Minimizing Weighted Tardiness/Earliness and Flow Time through Genetic Algorithm. *Journal of Optimization in Industrial Engineering*, 7(14):61-73.

Babu, S. and Girish, B. (2025). Neighborhood search-based metaheuristics for the bi-objective Pareto optimization of total weighted earliness-tardiness and makespan in a JIT single machine scheduling problem. *Operations Research Perspectives*, 14:100335.

Bank, J. and Werner, F. (2001). Heuristic algorithms for unrelated parallel machine scheduling with a common due date, release dates, and linear earliness and tardiness penalties. *Mathematical and Computer Modeling*, 33(4):363-383.
Beyranvand, M.S., Peyghami, M.R., and Ghatee, M. (2012). On the quadratic model for the unrelated parallel machine scheduling problem with a restrictive common due date. *Optimization Letters*, 6(8):1897-1911.

Caricato, P., Grieco, A., Arigliano, A. and Rondone, L. (2021). Workforce influence on manufacturing machines' schedules. *The International Journal of Advanced Manufacturing Technology*, 115:915-925.

Cheng, C.Y. and Huang, L.W. (2017). Minimizing total earliness and tardiness through unrelated parallel machine scheduling using distributed release time control. *Journal of Manufacturing Systems*, 42:1-10.

Cheng, T.C.E. (1989). A Heuristic for Common Due-date Assignment and Job Scheduling on Parallel Machines. *Journal of the Operational Research Society*, 40(12):1129-1135.

Chen, Z.L. and Powell, W.B. (1999). A column generation-based decomposition algorithm for a parallel machine just-in-time scheduling problem. *European Journal of Operational Research*, 116(1):220-232.

Eiben, A.E. and Smith, J.E. (2015). *Introduction to Evolutionary Computing*. Springer, Berlin, Heidelberg.

Ekici, A., Elyasi, M., Özener, O.O. and Sarıkaya, M.B. (2019). An application of unrelated parallel machine scheduling with sequence-dependent setups at Vestel Electronics. *Computers & Operations Research*, 111:130-140.

Fan, J., Zhang, C., Shen, W., and Gao, L. (2023). A matheuristic for the flexible job shop scheduling problem with lot-streaming and machine reconfigurations. *International Journal of Production Research*, 61(19): 6565-6588.

Foroutan, R.A, Shafipour, M., Rezaeian, J., and Khojasteh, Y., 2024. Just-in-time scheduling of unrelated parallel machines with family setups and soft time window constraints. *Journal of Industrial and Production Engineering*, 41(8):692-715.

Gordon, V., Proth, J-M., and Chu, C. (2002). A survey of the state-of-the-art of common due date assignment and scheduling research. *European Journal of Operational Research*, 139(1):1-25.

Grieco, A., Caricato, P. and Arigliano, A. (2022). A production planning and scheduling problem focused on both productivity and quality issues in tannery industries. *Procedia CIRP*,112:573-578.

Janiak, A., Janiak, W.A., Krysiak, T. and Kwiatkowski, T. (2015). A survey on scheduling problems with due windows. *European Journal of Operational Research*, 242(2):347-357.

Kayvanfar, V., Komaki, G.M., Aalaei, A. and Zandieh, M. (2014). Minimizing total tardiness and earliness on unrelated parallel machines with controllable processing times. *Computers & Operations Research*, 41(1):31-43.

Kim, J.G., Kim, J.S., and Lee, D.H. (2012). Fast and meta-heuristics for common due-date assignment and scheduling on parallel machines. *International Journal of Production Research*, 50(20):6040-6057.

Kramer, A. and Subramanian, A. (2019). A unified heuristic and an annotated bibliography for a large class of earliness-tardiness scheduling problems. *Journal of Scheduling*, 22(1):21-57.

Lin, S.W., Ying, K.C., Chiang, Y.I., and Wu, W.J. (2016). Minimizing total weighted earliness and tardiness penalties on identical parallel machines using a fast ruin-and-recreate algorithm. *International Journal of Production Research*, 54(22):6879-6890.

Li, X. and Gao, L. (2016). An effective hybrid genetic algorithm and tabu search for the flexible job shop scheduling problem. *International Journal of Production Economics*, 174:93-110.

M'hallah, R. and Al-Khamis, T. (2012). Minimizing total weighted earliness and tardiness on parallel machines using a hybrid heuristic. *International Journal of Production Research*, 50(10):2639-2664.

Mor, B., Mosheiov, G., and Shabtay, D. (2025). Scheduling problems on parallel machines with machine-dependent generalized due-dates. *Annals of Operations Research*, 347:1455-1471.

Mota, A., Ávila, P., Bastos, J., Roque, L.A.C., and Pires, A. (2025). Comparative Analysis of Simulated Annealing and Tabu Search for Parallel Machine Scheduling. *Procedia Computer Science*, 256:573-582.

Palhau, M., Sá, J.C., Ávila, P., Dinis-Carvalho, J., Rodrigues, C., Santos, G. (2024). Toyota Way - the Heart of Toyota Production System and its Impact on Sustainable Company Growth. *Quality Innovation Prosperity*, 28(3):23-45.

Pinedo, M.L. (2022). *Scheduling: Theory, Algorithms, and Systems*, 6th ed. Springer.

Plateau, M.C. and Rios-Solis, Y.A. (2010). Optimal solutions for unrelated parallel machines scheduling problems using convex quadratic reformulations. *European Journal of Operational Research*, 201(3):729-736.

Polyakovskiy, S. and M'Hallah, R. (2014). A multi-agent system for the weighted earliness-tardiness parallel machine problem. *Computers & Operations Research*, 44:115-136.

Putnik, G. and Ávila, P. (2016). Governance and Sustainability (Special Issue Editorial). *International Journal of Industrial and Systems Engineering*, 24(2):137-143.

Rabbani, M., Alipour, F., Farrokhi-Asl, H., and Manavizadeh, N. (2018). Using metaheuristic algorithms for solving a mixed model assembly line balancing problem considering express parallel line and learning effect. *Brazilian Journal of Operations & Production Management*, 15(2):254-269.

Rezaeian, J., Mohammad-Hosseini, S., Zabihzadeh, S., and Shokoufi, K. (2020). Fuzzy Scheduling Problem on Unrelated Parallel Machines in JIT Production System. *Artificial Intelligence Evolution*, 1(1):17-33.

Rios-Solis, Y.A. and Sourd, F. (2008). Exponential neighborhood search for a parallel machine scheduling problem. *Computers & Operations Research*, 35(5):1697-1712.

Rolim, G.A. and Nagano, M.S. (2020). Structural properties and algorithms for earliness and tardiness scheduling against common due dates and windows: A review. *Computers & Industrial Engineering*, 149:106803.

Snauwaert, J. and Vanhoucke, M. (2025). A solution framework for multi-skilled project scheduling problems with hierarchical skills. *Journal of Scheduling*, 28:289-310.

Sugimori, Y., Kusunoki, K., Cho, F., and Uchikawa, S. (1977). Toyota production system and Kanban system. Materialization of the just-in-time and respect-for-human system. *The International Journal of Production Research*, 15(6):553-564.

Sun, H. and Wang, G. (2003). Parallel machine earliness and tardiness scheduling with proportional weights. *Computers & Operations Research*, 30(5):801-808.

T'kindt, V., Shang, L. and Della Croce, F. (2020). Exponential time algorithms for just-in-time scheduling problems with common due date and symmetric weights. *Journal of Combinatorial Optimization*, 39(3):764-775.

Zarandi, M.H.F. and Kayvanfar, V. (2015). A bi-objective identical parallel machine scheduling problem with controllable processing times: a just-in-time approach. *The International Journal of Advanced Manufacturing Technology*,77:545-563.

Zhang, Y., Cheng, Z., Zhang, N., and Chiong, R. (2025). A weighted distribution-free model for parallel machine scheduling with uncertain job processing times. *European Journal of Operational Research*, 324:814-824.

APPENDIX A – MIXED-INTEGER LINEAR PROGRAMMING FORMULATION USED FOR PERFORMANCE EVALUATION

This section presents a MILP formulation employed to compute optimal or near-optimal reference solutions for small problem instances. These solutions support the convergence and performance analysis of the proposed metaheuristics. The appendix includes definitions of sets, parameters, and decision variables, as well as the mathematical formulation.

A.1 Sets and Indices

$J = \{1, \dots, n\}$: set of jobs
$M = \{1, \dots, m\}$: a set of machines
$R = \{1, \dots, n\}$: a set of positions on each machine

A.2 Parameters

p_{jk}: processing time of job j on machine k.
r_j: release time of job j.
d_j: due date of job j.
α_j: earliness weight of job j.
β_j: tardiness weight of job j.
M: sufficiently large positive constant.

A.3 Decision Variables

$x_{jkr} \in \{0,1\}$: equals 1 if job j is assigned to position r on machine k; 0 otherwise.
S_{kr}: start time of position r on machine k.
C_{kr}: completion time of position r on machine k.
c_j: completion time of job j.
E_j: earliness of job j.
T_j: tardiness of job j.

A.4 Mathematical Formulation

$$\min \quad \sum_{j=1}^{n} \alpha_j E_j + \beta_j T_j \tag{16}$$

$$s.t. \quad \sum_{k=1}^{m} \sum_{r=1}^{n} x_{jkr} = 1 \, , j \in J \tag{17}$$

$$\sum_{j=1}^{n} x_{jkr} \leq 1, k \in M, r \in R \tag{18}$$

$$\sum_{j=1}^{n} x_{jkr} \leq \sum_{j=1}^{n} x_{jk(r+1)}, k \in M, r \in R \setminus \{n\} \tag{19}$$

$$C_{kr} \geq S_{kr} + p_{jk} x_{jkr} - M(1 - x_{jkr}), j \in J, k \in M, r \in R \tag{20}$$

$$C_{kr} \leq S_{kr} + p_{jk} x_{jkr} + M(1 - x_{jkr}), j \in J, k \in M, r \in R \tag{21}$$

$$C_{k(r+1)} \geq C_{kr} + \sum_{j=1}^{n} p_{jk} x_{jk(r+1)}, k \in M, r \in R \setminus \{n\} \tag{22}$$

$$S_{kr} \geq r_j x_{jkr}, j \in J, k \in M, r \in R \tag{23}$$

$$c_j \geq C_{kr} + p_{jk} x_{jkr} - M(1 - x_{jkr}), j \in J, k \in M, r \in R \tag{24}$$

$$c_j \leq C_{kr} + p_{jk} x_{jkr} + M(1 - x_{jkr}), j \in J, k \in M, r \in R \tag{25}$$

$$c_j + E_j - T_j = d_j, \quad j \in J \tag{26}$$

$$x_{jkr} \in \{0,1\}, j \in J, k \in M, r \in R \tag{27}$$

$$C_{kr}, S_{kr}, \geq 0, \quad k \in M, \quad r \in R \tag{28}$$

$$c_j, E_j, T_j \geq 0, j \in J \tag{29}$$

Equation (16) defines the objective function, which minimizes the total weighted earliness and tardiness of all jobs. Equation (17) ensures that each job is assigned exactly once to a machine and a position. Equation (18) imposes a capacity constraint, allowing at most one job to be processed at each machine position. Equation (19) eliminates internal idle gaps by enforcing consecutive job assignments on each machine.

Equations (20) and (21) link the start and completion times of each position to the job's processing time assigned to that position. Equation (22) enforces the processing order of jobs on each machine, ensuring that positions are processed sequentially. Equation (23) guarantees that no job starts before its release time.

Equations (24) and (25) define the completion time of each job based on the completion time of the position to which it is assigned. Equation (26) establishes the relationship between job completion times, earliness, and tardiness with respect to due dates. Finally, Equations (27) – (29) specify the domains of the decision variables.