*Proceedings of the International Conference on Industrial Engineering and Operations Management*

Publisher: IEOM Society International, USA DOI: 10.46254/IN05.20250097
Published: November 6, 2025

# Hybrid Metaheuristics for Pareto-based Bi-objective Optimization in a JIT Unrelated Parallel Machine Scheduling Problem

**Sona Babu and B. S. Girish**
Department of Aerospace Engineering
Indian Institute of Space Science and Technology
Valiamala, Thiruvananthapuram, Kerala, India
sonababu.sct@gmail.com, girish@iist.ac.in

## Abstract

This paper considers the simultaneous optimization of makespan and total weighted earliness-tardiness in an unrelated parallel machine scheduling problem in a just-in-time manufacturing environment, with distinct due windows, machine eligibility constraints and sequence-dependent setup times, permitting idle times in the schedules. Two hybrid metaheuristics are proposed to tackle the NP-hard problem. The paper presents an exact method for generating a piecewise linear convex trade-off curve between the objectives for a particular job sequence. The Pareto front of the trade-off curves obtained for multiple job sequences, generated by the hybrid metaheuristics, is constructed using a method from the literature. The comparative performance evaluation reveals that the proposed hybrid multi-objective particle swarm optimization – local search (MOPSO-LS) algorithm shows superior performance for smaller problem instances, and the proposed hybrid Pareto archived multi-objective cuckoo search – local search (PAMOCS-LS) algorithm shows superior performance for larger problem instances.

## Keywords

Metaheuristics, Makespan, Total weighted earliness-tardiness, Pareto optimization, Parallel machine scheduling

## 1. Introduction

Parallel machine scheduling problem (PMSP) is one of the extensively researched machine scheduling problems in the literature (Sterna, 2021; Ying et al., 2024). PMSPs are classified into three categories, namely identical PMSP, non-identical or uniform PMSP and unrelated PMSP (Ying et al., 2024). The findings of a cluster analysis of the literature on PMSP, performed by Ying et al. (2024), revealed that the majority of the research in PMSP focused on unrelated parallel machines, emphasizing the significance of the problem in today's industry. The unrelated parallel machine scheduling problem (UPMSP) is also the closest to real-world manufacturing environments and is complex to solve (Wang et al., 2023). In this paper, we have considered a UPMSP in a just-in-time (JIT) manufacturing environment.

Machine scheduling problems in JIT manufacturing systems involve assigning optimal completion times to a set of jobs that are to be delivered at their associated due dates or due windows, the deviations from which incur earliness and tardiness penalties (Babu and Girish, 2025). The necessity for a trade-off between earliness and tardiness penalties resulted in numerous works in the literature on the simultaneous optimization of earliness and tardiness in JIT-UPMSP (Sterna, 2021), most of which are based on the weighted sum optimization method. Assigning weights to earliness and tardiness objectives according to their respective priorities in the weighted sum optimization approach has resulted in the total weighted earliness-tardiness (TWET) objective, the minimization of which is one of the objectives considered in this paper. Optimizing TWET in scheduling problems results in scheduling the jobs closer to their earliest due dates with idle times inserted in the schedule, resulting in the jobs forming clusters in the schedule (Girish

et al., 2022). The time of completion of the last job in the schedule, across all the machines in a PMSP, termed makespan, is another scheduling objective widely discussed in the UPMSP (Ying et al., 2024). Though the minimization of TWET ensures JIT production in the system, makespan has to be minimized in conjunction with TWET to ensure the efficient utilization of available resources and better throughput (Babu and Girish, 2025). Most works in the literature on the simultaneous optimization of makespan and TWET in UPMSP, have used the weighted-sum optimization approach, employing heuristics and metaheuristics to solve the problem (Đurasević and Jakobović, 2023). The weighted sum optimization approach provides a single optimal solution for each job sequence corresponding to the priorities for the objectives, which are assigned as weights. Therefore, the weighted-sum optimization approach requires the end users to specify their preferences for the objectives prior to scheduling (Neufeld et al., 2023). In contrast, the Pareto-based optimization approach provides several optimal solutions on a Pareto front. This allows the end users to choose a solution from the Pareto solution set, according to their priorities for the objectives (Neufeld et al., 2023). However, there exists no work in the literature on the simultaneous optimization of makespan and TWET in UPMSP using Pareto-based approaches, except Azevedo et al. (2023). In this paper, we have addressed the simultaneous optimization of makespan and TWET using a Pareto-based optimization approach.

The UPMSP has also been widely researched for the multi-objective optimization of several other objectives, considering additional parameters and constraints, viz., machine eligibility constraints, sequence-dependent setup times (SDST), etc. (Ying et al., 2024). It is evident from the literature that most works on the simultaneous optimization of makespan and TWET in the UPMSP have employed heuristics and metaheuristics (Ying et al., 2024). This is because the UPMSP is strongly NP-hard, even without considering the additional parameters and constraints (Đurasević and Jakobović, 2023). This paper considers the simultaneous optimization of makespan and TWET in a UPMSP with distinct due windows, machine eligibility constraints and sequence-dependent setup times, permitting idle times in the schedules, which is clearly NP-hard. We have adapted two population-based metaheuristic algorithms, namely the Pareto archived multi-objective cuckoo search (PAMOCS) algorithm and the multi-objective particle swarm optimization (MOPSO) algorithm, and hybridized each of them with a local search (LS) procedure that generates multiple job sequences. A job sequence may result in a piecewise linear convex trade-off curve between makespan and TWET since the problem permits inserting idle times into the schedules, similar to the works in the literature (Babu and Girish, 2024, 2025; Jacquin et al., 2018). No existing studies on the Pareto-based optimization of makespan and TWET in UPMSP have considered the insertion of idle times into the schedules. We have therefore adapted an exact method from the literature (Babu and Girish, 2025) to generate piecewise linear convex trade-off curves between makespan and TWET for the sequences of jobs generated by the hybrid metaheuristics. The Pareto front of the trade-off curves obtained for multiple sequences of jobs, generated using an exact Pareto front generation method adopted from the literature (Babu and Girish, 2024), which is also composed of line segments and points, provides the end users with a Pareto solution set from which they can choose the solutions that best satisfy their preferences and priorities. This is the first study to present an exact timing algorithm for the generation of all possible trade-offs between makespan and TWET for a given sequence of jobs in a UPMSP, allowing the insertion of idle times in the schedule. This is also the first study to present hybrid population-based metaheuristics for a bi-objective UPMSP in a JIT production environment, generating a Pareto front composed of line segments and points.

The subsequent sections of the paper are structured as follows. Section 2 discusses the mathematical formulation of the problem. Section 3 discusses the proposed exact method for the generation of the trade-off curve between makespan and TWET. Section 4 discusses the proposed hybrid metaheuristics. Section 5 discusses the performance evaluation of the proposed hybrid metaheuristics. Section 6 discusses the practical implications, and Section 7 concludes with the possible avenues for future research.

## 2. Mathematical formulation of the problem

The UPMSP addressed in this paper assumes $n$ jobs released simultaneously to be processed non-preemptively on $m$ parallel machines. Let $i, j (i, j = 1,2, \dots, n)$ denote the job index, and $k (k = 1,2, \dots, m)$ denote the machine index. Each job $i$ requires processing exactly one operation on a machine, and the set of all the machines eligible for processing job $i$ is denoted by $M_i$. Let $P_{ki}$ denote the processing time of job $i$ on machine $k: k \in M_i$. Let $S_{kij}$ denote the setup time to switch to job $j$ from job $i$ on machine $k$. Let $[de_i, dt_i]$ denote the due window of job $i$, where $de_i$ and $dt_i$ respectively represent the earliest and latest due dates. All job descriptors are predefined and deterministic, and all the machines are continuously available. Let $C_i$ denote the completion time of job $i$. Then, the earliness is expressed as $E_i = max(0, de_i - C_i)$, and the tardiness is expressed as $T_i = max(0, C_i - dt_i)$. Let $\alpha_i$ and $\beta_i$,

respectively, denote the penalties imposed for early and tardy completion of job *i*. The mathematical representation of the problem is as follows (Nogueira et al., 2014).

Input Parameters:
$M_i$: Set of all the machines eligible for processing job *i*
$J_k$: Set of all the jobs eligible for allocation on machine *k*, including the dummy job 0

Decision Variables:
$X_{kij} = \begin{cases} 1: \text{if job } j \text{ succeeds job } i \text{ on machine } k \\ 0: \text{otherwise} \end{cases}$
$C_i = \text{completion time of job } i$
$E_i = \text{earliness of job } i$
$T_i = \text{tardiness of job } i$

Objective:
Minimize $\sum_{i=1}^{n}(\alpha_i E_i + \beta_i T_i)$ (1)
Minimize $max_i\ (C_i)$ (2)
Subject to:
$\sum_{j\in J_k} X_{k0j} = 1\ \ \forall k$ (3)
$\sum_{i\in J_k} X_{ki0} = 1\ \ \forall k$ (4)
$\sum_{k\in M_i}\sum_{j\in J_k: j\neq i} X_{kij} = 1\ \ \forall i: i = 1,2,\dots,n$ (5)
$\sum_{k\in M_j}\sum_{i\in J_k: i\neq j} X_{kij} = 1\ \ \forall j: j = 1,2,\dots,n$ (6)
$\sum_{i\in J_k: i\neq j} X_{kij} = \sum_{l\in J_k: l\neq j} X_{kjl}\ \ \forall k, j: j \in J_k\ \&\ j \neq 0$ (7)
$C_j \geq \sum_{k\in M_j}(P_{kj}.X_{k0j}), \forall j: j = 1,2,\dots,n$ (8)
$C_j \geq C_i - M + (P_{kj} + S_{kij} + M)\, X_{kij}\ \ \forall k, i, j: i \in J_k\ \&\ j \in J_k, i \neq 0, j \neq 0$ (9)
$T_i \geq C_i - dt_i\ \ \forall i, k$ (10)
$E_i \geq de_i - C_i\ \ \forall i, k$ (11)
$C_{max} \geq C_i\ \ \forall i:\ i = 1,2,\dots,n$ (12)
$X_{kij} \in \{0,1\}\ \ \forall k, i, j: i \in J_k, j \in J_k, i \neq j$ (13)
$E_i \geq 0, T_i \geq 0\ \ \ \forall i: i = 1,2,\dots,n$ (14)

Constraint (3) ensures there is a dummy job at the start of every machine and exactly one job succeeding it. Constraint (4) ensures there is a dummy job at the end of every machine and exactly one job preceding it. The activities succeeding and preceding the dummy activities in constraints (3) and (4), respectively, can be a dummy job, implying that no job is allocated to the machine. Constraint (5) guarantees that each job ***i*** is allocated to a unique machine and is succeeded by exactly one job ***j***, which can be a dummy job. Constraint (6) guarantees that each job ***j*** is allocated to a unique machine and is preceded by exactly one job ***i***, which can be a dummy job. Constraint (7) guarantees that a job is allocated to a unique machine, and that if a job ***j*** has a preceding job ***i***, then ***j*** also has a succeeding job ***l***, where ***i*** and ***l*** can be dummy activities. Constraint (8) guarantees that the completion time of the first job on each machine is at least equal to its processing time. Constraint (9) associates the completion times of a job ***j*** and its preceding job ***i*** on a machine ***k***. ***M*** denotes a large positive integer. Constraints (10) and (11) respectively relate the tardiness and earliness of each job with its completion time and due windows. Constraint (12) establishes the maximum completion time. Constraints (13) and (14) establish the variable bounds.

## 3. Proposed exact algorithm for the TWET-makespan trade-off curve generation

The proposed metaheuristics represent a solution as a permutation of job indices, indicating the sequences of jobs allocated for processing on each of the unrelated parallel machines in the scheduling environment. Let $\mu$ ($\mu = \{1,2,\dots,m\}$) represent the set of $m$ parallel machines and $\sigma$ ($\sigma = \{\sigma_1, \sigma_2, \dots, \sigma_m\}$) represent a set of job sequences allocated to each of the machines, where each job sequence $\sigma_p$ represents an ordered set of $n_p$ jobs allocated to the machine $p\ \forall p \in \mu$. Let $\sigma_p[k]$ denote the job identifier at the $k^{th}$ position index of $\sigma_p$. The generation of the trade-off curve between makespan and TWET, for a given $\sigma$, begins by generating the TWET-optimal schedule, and subsequently optimizing its makespan iteratively. The generation of the TWET-optimal schedule for a given $\sigma$ in PMSP is equivalent to separately generating the TWET-optimal schedules $\forall\sigma_p \in \sigma$ on the corresponding $p$ based on

exact algorithms discussed in the literature for a single machine scheduling problem (SMSP) (Babu and Girish, 2025). The TWET-optimal schedule, hence generated for the UPMSP, denoted by $C$ ($C = \{C_1, C_2, \ldots, C_n\}$), is input to Algorithm 1, and the corresponding values of TWET and makespan are saved as the first trade-off point, as shown in steps 1-2. The time of completion of the last job on the machines with the maximum makespan in schedule $C$ are then iteratively left shifted, generating the remaining trade-off points. This is implemented by suitably adapting an exact method of trade-off curve generation presented by Babu and Girish (2025) for an SMSP to the UPMSP, as shown in Algorithms 1 and 2.

**Algorithm 1: TWET-makespan trade-off curve generation procedure**

```
    Data: σ, μ, de_i, dt_i, α_i, β_i, ∀i ∈ σ, n_k ∀k, P_ki ∀k, i, S_kii' ∀ k, i, i': k ∈ μ, i, i' ∈ σ_k, i ≠ i'
1   C ← Optimal TWET schedule, t ← 1, PSL ← 0
2   SAVE_BREAK_POINT(t)
3   while (t_idle! = 0) do
4       for (∀p ∈ p_max) do        ▷ p_max is the set of machines with makespan equal to the maximum makespan
5           B_p ← ∅
6           if (n_p = 1)
7               B_p ← {σ_p[n_p]}
8           else
9               for (i = n_p to 2) do
10                  if (C_σp[i−1] + P_pσp[i] + S_pσp[i−1]σp[i] = C_σp[i]) then
11                      B_p ← B_p ∪ {σ_p[i]}
12                  else
13                      break
14                  end
15              end
16          end
17          SL_p ← 0
18          for (∀j ∈ B_p) do
19              if (C_σp[j] ≤ de_σp[j]) then
20                  SL_p ← SL_p − α_σp[j]
21              else if (C_σp[j] > dt_σp[j]) then
22                  SL_p ← SL_p + β_σp[j]
23              end
24          end
25      end
26      OSL ← Σ_{p∈p_max} SL_p
27      if (OSL ≠ PSL) then
28          t ← t + 1, PSL ← OSL
29      end
30      C ← LEFT_SHIFT(p_max)
31  end
32  Function SAVE_BREAK_POINT(t)
33      G_t ← Σ_{i=1}^{n} (α_i max{0, de_i − C_i} + β_i max{0, C_i − dt_i})
34      M_t ← max_{q∈μ} {max_{1≤i≤n_q} {C_σ[i]}}
35  end
```

In Algorithm 1, $p_{max} \in \mu$ represents the set of machines with the maximum makespan in schedule $C$ at a given time, and the time of completion of the last job on every $p \in p_{max}$ are simultaneously left shifted. Let $\sigma_p[k-1]$ and $\sigma_p[k]$ represent two successive jobs on a machine $p$. If the last job $\sigma_p[n_p]$ on any $p \in p_{max}$ has contiguously scheduled jobs preceding it, as implied by the equality condition in step 10, the contiguous jobs are left shifted together as a block, denoted by $B_p$. Steps 5-16 identify the block of jobs $B_p$ to be left-shifted on $p$ $\forall p \in p_{max}$. Steps 17-24 calculate the

TWET cost function slope of block $B_p$, denoted by $SL_p\ \forall p \in p_{max}$ (Girish et al., 2022). The cost function slope contributed by the blocks on all the machines belonging to $p_{max}$, denoted by $OSL$, is then calculated, and the identified $B_p \forall p \in p_{max}$ is passed to the function *LEFT_SHIFT*, described in Algorithm 2.

**Algorithm 2: Left shifting procedure**

1 **Function** $LEFT\_SHIFT(p_{max})$
2 $\delta \leftarrow M,\ \delta_p \leftarrow M\ , l_p \leftarrow |B_p|\ \forall p \in p_{max}$ ▷ $M$ is a large positive integer
3 **for** $(\forall p\ \in p_{max})$ **do**
4 $t_i \leftarrow 0\ \forall i \in \{1,2,3,4\}$
5 **if** $(\sigma_p[1] \in B_p)$ **then**
6 $\delta_p \leftarrow C_{\sigma_p[1]} - P_{\sigma_p[1]}$
7 **end**
8 **if** $(\delta_p\ >\ 0)$ **then**
9 **if** $(l_p\ <\ n_p)$ **then**
10 $a \leftarrow\ \sigma_p[n_p - l_p], b \leftarrow \sigma_p[n_p - l_p + 1]$
11 $t_1 \leftarrow C_b - P_{pb} - S_{pab} - C_a$
12 **end**
13 $t_2 \leftarrow \min_{i \in B_p} \left\{C_{\sigma_p[i]}\ -\ dt_{\sigma_p[i]}\ :\ C_{\sigma_p[i]}\ >\ dt_{\sigma_p[i]}\right\},\ t_3\ \leftarrow\ \min_{i \in B_p} \left\{C_{\sigma_p[i]}\ -\ de_{\sigma_p[i]}\ :\ de_{\sigma_p[i]} < C_{\sigma_p[i]} \leq\ dt_{\sigma_p[i]}\right\}$
14 $t_4 \leftarrow \min_{r \in \mu: r \notin p_{max}} \left\{C_{\sigma_p[n_p]} - C_{\sigma_r[n_r]}\right\}$
15 $\delta_p \leftarrow \min\{\delta, \min\{t_i : i \in \{1,2,3,4\}, t_i > 0\}\}$
16 **else**
17 $\delta \leftarrow 0$
18 break
19 **end**
20 **end**
21 **if** $(\delta != 0)$ **then**
22 $\delta \leftarrow \min_{p \in p_{max}} (\delta_p)$
23 **for** $(\forall p\ \in p_{max})$ **do**
24 $C_{\sigma_p[j]} \leftarrow C_{\sigma_p[j]} - \delta\ \ \forall j \in B_p$
25 **end**
26 $SAVE\_BREAK\_POINT(t)$
27 **else**
28 $t_{idle} \leftarrow 0$
29 **end**
31 **end**

Let $\delta$ represent the maximum units of time by which the jobs in $B_p\ \forall p \in p_{max}$ can be simultaneously left shifted and $\delta_p$ represent the maximum units of time by which the jobs in each $B_p$ can be left shifted on the corresponding $p \in p_{max}$ subject to non-overlapping constraints on job completion times for the jobs in $\sigma_p$. For each $p \in p_{max}$, $\delta_p$ is the maximum time units of left shifting possible until either no idle time is available before the last job in $B_p$ in the corresponding $\sigma_p$, or the completion time of a job in $B_p$ either becomes equal to its latest due date or earlier than its earliest due date or equal to the completion time of the last job on any machine $p' \in \mu$: $p' \notin p_{max}$, on left shifting. This is as shown in steps 3-20. In scenarios where $p_{max}$ comprises multiple machines, $\delta$ is assigned the smallest $\delta_p$ among $p\ \forall p \in p_{max}$, as shown in step 22. The completion times of jobs in $B_p \forall p \in p_{max}$ are simultaneously left shifted by $\delta$ time units, and the TWET and makespan values of the left shifted schedule, denoted by $G_t$ and $M_t$, are saved as the subsequent breakpoint using the function $SAVE\_BREAK\_POINT$ shown in steps 32-35 of Algorithm 1. If the cost function slope, $OSL$, varies with left shifting $(i.e., OSL \neq PSL)$, a new breakpoint is generated on the trade-off curve, and the breakpoint identifier $t$ is incremented by one, as shown in steps 26-29 of Algorithm 1. However, if $OSL = PSL$, the break point that is stored at index $t$, in step 26 of Algorithm 2, is overwritten at the end

of the subsequent iteration of left shift. Steps 4-30 of Algorithm 1 are repeated until the idle time preceding the first job $\sigma_p[1]$ on one of the machines in $p_{max}$ is eliminated, i.e., $\delta_p = 0$ for any $p \in p_{max}$. The breakpoints obtained are subsequently joined with line segments, resulting in the optimal piecewise linear convex trade-off curve between makespan and TWET, which is effectively the Pareto-optimal front for the sequence $\sigma$ (Babu and Girish, 2025).

## 4. Proposed hybrid metaheuristics

This section presents the proposed hybrid metaheuristics, namely the hybrid Pareto archived multi-objective cuckoo search – local search (PAMOCS-LS) algorithm and the hybrid multi-objective particle swarm optimization local search (MOPSO-LS) algorithm. The local search employs pairwise swap and insertion neighbourhoods, denoted by $N_1$ and $N_2$, respectively, applied both within and across different machines.

### 4.1 Initial solution construction

The initial solution set for the proposed hybrid metaheuristics was generated by suitably adapting the initial solution generation procedure described by Babu and Girish (2025) for an SMSP. The construction of initial solutions for the UPMSP begins with an empty set $\sigma$ with $m$ subsets, to which $n$ unscheduled jobs are sequentially added according to a probabilistic rule guided by heuristic desirability. Each subset $\sigma_y \in \sigma: y \in \{1,2,\dots,m\}$ denotes the partial sequence on one of the machines in the UPMSP. The heuristic desirability $\eta_{ykb}$ of assigning a job $b$ to position $k$ in the $y^{th}$ subset of $\sigma$ is defined as

$$\eta_{ykb} = exp\left[-\frac{de_b * k_1}{d_{avg}}\right] * exp\left[-\frac{(\sum_{j\in\sigma_y} P_{yj} + P_{yb} + S_{y_{ab}}) * k_2}{PS_{y_{avg}}}\right] \quad (15)$$

where $de_b$ denotes the earliest due date of the job $b$, $d_{avg}$ denotes the average of the earliest due dates of the jobs unassigned to $\sigma$, $k_1$ and $k_2$ denote the scaling parameters associated with the due dates and the sum of processing times and setup times ($P_{y_b} + S_{y_{ab}}$), respectively. $P_{yb}$ denotes the processing time of the job $b$ on machine $y$, and $\sum_{j\in\sigma_y} P_{yj}$ denotes the sum of the processing times of all the jobs in the partial sequence $\sigma_y$. $S_{y_{ab}}$ denotes the setup time between jobs $b$ and its preceding job in $\sigma_y$, denoted by $a$, and $PS_{y_{avg}}$ denotes the average of the processing times and setup times between the jobs unassigned to $\sigma_y$. The job $j$ to be assigned at each position in $\sigma_y \in \sigma: y \in \{1,2,\dots,m\}$ is selected probabilistically based on the value of the random variable $S$ drawn according to a probability $\mu_{ykb}$ defined for each unassigned job $b \in U$, as shown in Eq. (16).

$$\mu_{ykb} = \frac{\eta_{ykb}}{\sum_{u\in U}(\eta_{yku})} \quad (16)$$

where $U$ denotes the set of jobs not assigned to $\sigma$. For each position $k$ in $\sigma$, the cumulative value of $\mu_{ykb} \forall b \in U$ is evaluated, and the job $b$ corresponding to the interval of the random variable $S$ is assigned to $k$. This process iterates until all the $n$ jobs are appended to some $\sigma_y \in \sigma$. The sequences hence constructed comprise the initial solution set, denoted by $P$, in the population-based metaheuristic algorithms presented in the following sections.

### 4.2 The hybrid PAMOCS-LS algorithm

Cuckoo search is a population-based metaheuristic algorithm inspired by the brood parasitism behaviour of cuckoo birds. A cuckoo represents a candidate solution on which one or more search strategies are applied to generate new candidate solutions, termed eggs. The eggs laid by each cuckoo are placed in a habitat that the cuckoo randomly picks from the solution space, termed a nest. The eggs in the selected nest, along with the newly laid ones, are evaluated for objective values to preserve the good ones in Pareto archives and discard the bad ones, analogous to host birds rejecting foreign eggs. The proposed hybrid PAMOCS-LS is inspired by the PAMOCS algorithm presented by Nartu et al. (2019) for a multi-objective minimization problem, where each candidate solution results in a single trade-off point on the Pareto front. In this paper, we have suitably extended their methodology for a multi-objective UPMSP where each job sequence results in multiple line segments on the Pareto front. The proposed hybrid PAMOCS-LS, shown in Algorithm 3, begins by generating the trade-off curves between makespan and TWET $\forall \sigma \in P$, using the exact algorithm presented in Section 3, and subsequently constructing the global Pareto front of the trade-off curves using a method adopted from the literature (Babu and Girish, 2024), denoted by $A$, as shown in steps 1-2. Nests denoted by $Nest_p$ corresponding to each $\sigma \in P$ are initialized with the respective $\sigma$ as the first egg, the set of which is denoted by $N$, where $p$ is the nest identifier. Local Pareto fronts of the solutions belonging to each nest $p$ are also generated,

denoted by $A_{Nest_p} \forall \sigma \in P$, as shown in steps 4-8. The search for optimal solutions begins with randomly selecting a candidate solution, denoted as $Cuckoo$, from a randomly selected nest $p' \in N$, and subjecting it to $N_2$ randomly to generate $k$ new candidate solutions, the set of which is denoted as $Eggs$. The non-dominated candidate solutions in set $Eggs$, belonging to the Pareto front $A_{Eggs}$, update the local Pareto front $A_{Nest_p}$ corresponding to the randomly selected nest $p \in N$. This is as shown in steps 10-16. The solutions belonging to $A_{Nest_p}$ are further subjected to local search using $N_1$ and $N_2$, as shown in steps 17-25. For every $\sigma \in A_{Nest_p}$, $visited\_flag(\sigma, N_k)\ \forall N_k \in \{N_1, N_2\}$ is a binary variable that ensures that $\sigma$ is subjected to local search using a particular $N_k$, only once. The solutions in the Pareto front $A'$ of the set of neighbourhoods belonging to set $M$ iteratively updates the local Pareto front $A_{Nest_p}$ corresponding to nest $p$, following which $A_{Nest_p}$ updates the global Pareto front $A$. The search procedure in steps 10-26 repeats until a predefined maximum computation time, denoted by $CPU\_TimeLimit$, after which the global Pareto front $A$ comprises the optimal solutions to the problem.

**Algorithm 3: The hybrid PAMOCS-LS algorithm**

1 $P \leftarrow InitialPopulationGeneration()$
2 $A \leftarrow ParetoOptimalFrontGeneration(P)$
3 $p \leftarrow 0,\ N \leftarrow 0, visited\_flag(\sigma, \{N_1, N_2\}) \leftarrow false\ \ \forall \sigma \in A$
4 **for** $(\forall \sigma\ \in P)$ **do**
5 $\quad p \leftarrow p + 1$
6 $\quad Nest_p \leftarrow \sigma, N \leftarrow N \cup \{Nest_p\}$
7 $\quad A_{Nest_p} \leftarrow ParetoOptimalFrontGeneration(Nest_p)$
8 **end**
9 **while** $(CPU\_Time < CPU\_TimeLimit)$ **do**
10 $\quad p' \leftarrow SelectNest(N)$
11 $\quad Cuckoo \leftarrow SelectJobSequence(A_{Nest_{p'}})$
12 $\quad Eggs \leftarrow GenerateRandomNeighbourhoods(Cuckoo, N_2)$
13 $\quad A_{Eggs} \leftarrow ParetoOptimalFrontGeneration(Eggs)$
14 $\quad visited\_flag(\sigma, \{N_1, N_2\}) \leftarrow false\ \ \forall \sigma \in A_{Eggs}$
15 $\quad p \leftarrow SelectNest(N)$
16 $\quad A_{Nest_p} \leftarrow ParetoOptimalFrontUpdate(A_{Nest_p} \cup A_{Eggs})$
17 $\quad$ **for** $(\forall N_k\ \in \{N_1, N_2\})$ **do**
18 $\quad\quad$ **while** $(visited\_flag(\sigma, N_k) \neq\ true\ \forall \sigma \in A_{Nest_p})$ **do**
19 $\quad\quad\quad M \leftarrow GenerateNeighbourhoods(\sigma, N_k)\ \ \forall \sigma \in A_{Nest_p}$: *visited_flag*$(\sigma, N_k)$=*false*
20 $\quad\quad\quad$ *visited_flag*$(\sigma, N_k) \leftarrow$ *true* $\forall \sigma \in A_{Nest_p}$
21 $\quad\quad\quad A' \leftarrow ParetoOptimal\ FrontGeneration(M)$
22 $\quad\quad\quad$ *visited_flag*$(\sigma, N_k) \leftarrow$ *false* $\forall \sigma \in A'$
23 $\quad\quad\quad A_{Nest_p} \leftarrow ParetoOptimalFrontUpdate(A_{Nest_p} \cup A')$
24 $\quad\quad$ **end**
25 $\quad$ **end**
26 $\quad A \leftarrow ParetoOptimalFrontUpdate(A \cup A_{Nest_p})$
27 **end**

## 4.3 The hybrid MOPSO-LS algorithm

Particle swarm optimization is a population-based metaheuristic algorithm inspired by the collective movement of birds towards a food source. Each candidate solution, termed a particle, has an associated position and velocity, which it updates while moving through the solution space, interacting with other particles, the group of which is termed swarm. The velocity and position of each particle are updated based on the personal best position in their own memory and the best position discovered by the swarm, thus gradually converging towards the optimal solutions. The proposed hybrid MOPSO-LS is inspired by the PSO algorithm presented by Girish and Jawahar (2009) for a single objective job shop scheduling problem, which we have suitably adapted for the bi-objective UPMSP. The proposed hybrid MOPSO-LS, is as shown in Algorithm 4.

**Algorithm 4: The hybrid MOPSO-LS algorithm**

1 $P \leftarrow InitialPopulationGeneration()$
2 $A \leftarrow ParetoOptimalFrontGeneration(P)$
3 $A_\sigma \leftarrow ParetoOptimalFrontGeneration(\sigma)\ \ \forall \sigma \in P$
4 $visited\_flag(\sigma, \{N_1, N_2\}) \leftarrow false\ \ \forall \sigma \in A$
5 **while** $(CPU\_Time < CPU\_TimeLimit)$ **do**
6  **for** $(\forall \sigma\ \in P)$ **do**
7  $k \leftarrow NearestLineSegment(\sigma, A)$
8  $\sigma_{Gbest} \leftarrow JobSequence(k, A)$
9  $k' \leftarrow NearestLineSegment(\sigma, A_\sigma)$
10  $\sigma_{Pbest} \leftarrow JobSequence(k', A_\sigma)$
11  $d_1 \leftarrow \{Insertion\ Moves\ on\ \sigma\ to\ reach\ \sigma_{p_{Gbest}}\}$
12  $d_2 \leftarrow \{Insertion\ Moves\ on\ \sigma\ to\ reach\ \sigma_{p_{Pbest}}\}$
13  **for** $(\forall d\ \in d_1)$ **do**
14  **if** $(randNum < gprob)$ **then** ▷ $randNum \in [0,1]$
15  $\sigma \leftarrow InsertionMove(\sigma, d_1)$
16  **end**
17  **end**
18  **for** $(\forall d\ \in d_2)$ **do**
19  **if** $(randNum < pprob)$ **then**
20  $\sigma \leftarrow InsertionMove(\sigma, d_2)$
21  **end**
22  **end**
23  **end**
24  **for** $(\forall \sigma\ \in P)$ **do**
25  $A_\sigma' \leftarrow ParetoOptimalFrontGeneration(\sigma)$
26  $A_\sigma \leftarrow ParetoOptimalFrontUpdate(A_\sigma \cup A_\sigma')$
27  **for** $(\forall N_k\ \in \{N_1, N_2\})$ **do**
28  **while** $(visited\_flag(\sigma, N_k) \neq\ true\ \forall \sigma \in A_\sigma)$ **do**
29  $M \leftarrow GenerateNeighbourhoods(\sigma, N_k)\ \ \forall \sigma \in A_\sigma$: *visited_flag*$(\sigma, N_k)$=*false*
30  *visited_flag*$(\sigma, N_k) \leftarrow$ *true* $\forall \sigma \in A_\sigma$
31  $A_\sigma'' \leftarrow ParetoOptimalFrontGeneration(M)$
32  *visited_flag*$(\sigma, N_k) \leftarrow$ *false* $\forall \sigma \in A_\sigma''$
33  $A_\sigma \leftarrow ParetoOptimalFrontUpdate(A_\sigma \cup A_\sigma'')$
34  **end**
35  **end**
36  $A \leftarrow ParetoOptimalFrontUpdate(A \cup A_\sigma)$
37  **end**
38 **end**

The proposed hybrid MOPSO-LS algorithm begins by constructing a global Pareto front of the solutions belonging to $P$, denoted by $A$, and local Pareto fronts corresponding to each $\sigma \in P$, denoted by $A_\sigma$, as shown in steps 1-3. The search for optimal solutions begins by identifying the global and personal best solutions corresponding to each $\sigma \in P$, denoted by $\sigma_{Gbest}$ and $\sigma_{Pbest}$, respectively, as shown in steps 7-10. $\sigma_{Gbest}$ and $\sigma_{Pbest}$ for each $\sigma \in P$ refer to the sequences associated with the nearest line segments, indexed $k$ and $k'$, on the respective Pareto fronts $A$ and $A_\sigma$ from the trade-off curve associated with $\sigma$. The nearest line segment on a Pareto front to the trade-off curve associated with a given $\sigma$, is obtained using the point of closest approach algorithm presented by Sunday (2006). Further, the insertion moves to transition $\sigma$ to $\sigma_{p_{Gbest}}$ and $\sigma_{p_{Pbest}}$ are identified and stored in sets $d_1$ and $d_2$, respectively. Sets $d_1$ and $d_2$ are ordered in descending order of the positional differences between the jobs in $\sigma$ and the jobs in $\sigma_{p_{Gbest}}$ and $\sigma_{p_{Pbest}}$, respectively, such that the insertion moves between machines are prioritized over the insertion moves within a machine. The insertion moves in sets $d_1$ and $d_2$ are then applied successively on $\sigma$ with the respective probabilities

$gprob$ and $pprob$, as shown in steps 13-22. The improved sequence $\sigma$ is then subjected to local search using $N_1$ and $N_2$, as shown in steps 27-35. The Pareto front $A_\sigma{''}$, of the neighbourhoods belonging to set $M$ iteratively updates the local Pareto front $A_\sigma$, following which $A_\sigma$ updates the global Pareto front $A$. The search procedure in steps 6-37 is performed until the $CPU\_TimeLimit$, after which the global Pareto front $A$ comprises the optimal solutions to the problem.

## 5. Performance evaluation

This section discusses the comparative performance evaluation of the proposed hybrid metaheuristics, implemented in C and executed using the Intel C++ Compiler v2022.2.1 on a Linux workstation featuring dual 2.6 GHz Intel Xeon Gold 6132 processors with 32 cores and 128 GB RAM. The neighbourhood generation procedure in the proposed metaheuristics was parallelized using OpenMP for efficient multi-core execution (Babu and Girish, 2025). The *-fp-model* flag was configured to *strict* during compilation to ensure floating point precision. The compiler optimization level was fixed at $-O0$, to disable possible optimization at runtime.

### 5.1 Test instance generation

As the literature presents no known benchmark data for the bi-objective UPMSP considered in this work, problem instances of different sizes were developed using the methodology described by Lee and Pinedo (1997). The number of jobs and machines in each problem instance, denoted by $\boldsymbol{n}$ and $\boldsymbol{m}$ respectively, are selected from the corresponding sets $\{\mathbf{20, 30, 40}\}$ and $\{\mathbf{3, 5}\}$ for small-sized instances, $\{\mathbf{50, 75, 100}\}$ and $\{\mathbf{4, 6, 8}\}$ for medium-sized instances, and $\{\mathbf{200, 300, 400}\}$ and $\{\mathbf{10, 20, 30}\}$ for large-sized instances. The processing time $\boldsymbol{P_{ki}}$, for a job $\boldsymbol{i}$ on a machine $\boldsymbol{k}$, was generated uniformly in $[\mathbf{50, 150}]$. The setup time $\boldsymbol{S_{kii'}}$ between two jobs $(\boldsymbol{i, i'})$: $\boldsymbol{i \neq i'}$ on each machine $\boldsymbol{k}$, was generated uniformly in $[\mathbf{1, 2\eta\overline{p}}]$ where $\boldsymbol{\eta}$ is the setup time severity factor and $\boldsymbol{\overline{P}}$ is the average processing time per job per machine, defined as $\boldsymbol{\overline{P} = (\sum_k \sum_i P_{ki}) / mn}$. The earliness and tardiness penalties, $\boldsymbol{\alpha_i}$ and $\boldsymbol{\beta_i}$, of each job $\boldsymbol{i}$ were generated uniformly in $[\mathbf{1, 100}]$. The due windows were generated uniformly in $[\boldsymbol{(1-R)\overline{d}, \overline{d}}]$ with a probability $\boldsymbol{\tau}$ and, in $[\boldsymbol{\overline{d}, \overline{d} + (C_{max} - \overline{d})R}]$ with a probability $\mathbf{1} - \boldsymbol{\tau}$, where $\boldsymbol{\tau}$ is the due date tightness factor, $\boldsymbol{R}$ is the due date range factor, $\boldsymbol{\overline{d}}$ is the average due date, and $\boldsymbol{C_{max}}$ is the makespan. $\boldsymbol{\overline{d}}$ and $\boldsymbol{C_{max}}$ are defined as $\boldsymbol{\overline{d} = C_{max}(1-\tau)}$ and $\boldsymbol{C_{max} = (\beta\overline{S} + \overline{P})\mu}$, respectively, where $\boldsymbol{\overline{S}}$ is the average setup time. The coefficient $\boldsymbol{\beta}$ accounts for the effect of setup time on makespan and is defined as $\boldsymbol{\beta = \left[0.4 + \frac{10}{\mu^2} - \frac{\eta}{7}\right]}$, where $\boldsymbol{\mu}$ is the job-machine factor defined as $\boldsymbol{\mu = n/m}$. The values of $\{\boldsymbol{\eta, \tau, R}\}$ were fixed at $\{\mathbf{0.2, 0.3, 0.25}\}$, respectively, based on trial-and-error. For each problem size category, $\boldsymbol{n * m}$ test instances were generated, which resulted in $\mathbf{3x2 + 3x3 + 3x3 = 24}$ problem instances that follow the naming convention $\boldsymbol{Psize\_Jn\_Mm}$: $\boldsymbol{Psize} \in \boldsymbol{(S, M, L)}$, where, $\boldsymbol{S, M}$ and $\boldsymbol{L}$, respectively, denote the small, medium and large size problem categories.

### 5.2 Optimal parameter settings

The parameters in the proposed hybrid metaheuristics were optimized by solving selected problem instances using the hybrid metaheuristics for ten runs. The optimal settings are listed below.

| Parameter | Optimal setting |
|---|---|
| Scaling parameters in initial solution generation | In PAMOCS-LS: $k_0 = 2; k_1 = 2; k_2 = 1$<br>In MOPSO-LS: $k_0 = 2; k_1 = 3; k_2 = 2$ |
| Number of eggs in PAMOCS-LS | $k = \begin{cases} 4, n < 100 \\ 6, 100 \leq n < 400 \\ 8, n \geq 400 \end{cases}$ |
| Probabilities in MOPSO-LS | $gprob = 0.4; pprob = 0.8$ |
| Termination criterion for all algorithms | $CPU\_TimeLimit = \begin{cases} 50 \text{ x } n \text{ seconds}, n \leq 40 \\ 85 \text{ x } n \text{ seconds}, n > 40 \end{cases}$ |

### 5.3 Performance comparison

This section compares the performance efficiency of the two hybrid metaheuristics using the hypervolume metric adopted from the literature (Babu and Girish, 2025). Table 1 shows the percentage deviations of the average hypervolume from the best average hypervolume computed for the Pareto fronts obtained by the proposed hybrid metaheuristics for 24 problem instances in ten runs. The percentage deviations in hypervolume close to 0 for an algorithm indicate its superior convergence over the other. Table 1 shows that the hybrid MOPSO-LS algorithm has superior convergence for most problem instances with up to 75 jobs, the two hybrid metaheuristic algorithms converge

competitively for problem instances with 100 to 200 jobs, and the hybrid PAMOCS-LS algorithm has superior convergence for all problem instances with 300 to 400 jobs. This is also evident from Figure 1 that shows the Pareto fronts obtained by the two hybrid metaheuristics for small and large-sized problem instances. The superior performance of the hybrid MOPSO-LS for smaller problem instances can be attributed to its large extent of exploitation of the solution space, resulting in faster convergence. The MOPSO algorithm performs a guided exploration of the solution space, and hybridizing the MOPSO algorithm with local search further enhances its exploitation. However, the hybrid MOPSO-LS algorithm tends to converge prematurely or get trapped in local optima for large-sized problems with large solution spaces due to insufficient exploration. The superior performance of the hybrid PAMOCS-LS for large problem instances can be attributed to its balanced exploitation and exploration of the solution space. The PAMOCS algorithm performs a random exploration of the solution space by Levy flights. Hybridizing it with local search balances its exploration with exploitation, leading to faster convergence for larger problem instances. However, for smaller problem instances, the increased exploration spreads the computational efforts over many new solutions, rather than exploiting the solution space, thereby performing inferior.

Table 1. Comparison of the hypervolume obtained by the proposed metaheuristics

| Sl. No. | No. of jobs | No. of machines | Problem instance | % Deviation in hypervolume | |
|---|---|---|---|---|---|
| | | | | PAMOCS-LS | MOPSO-LS |
| 1 | 20 | 3 | S_J20_M3 | 19.14 | 0 |
| 2 | 20 | 5 | S_J20_M5 | 8.10 | 0 |
| 3 | 30 | 3 | S_J30_M3 | 32.95 | 0 |
| 4 | 30 | 5 | S_J30_M5 | 15.41 | 0 |
| 5 | 40 | 3 | S_J40_M3 | 22.08 | 0 |
| 6 | 40 | 5 | S_J40_M5 | 2.07 | 0 |
| 7 | 50 | 4 | M_J50_M4 | 17.70 | 0 |
| 8 | 50 | 6 | M_J50_M6 | 5.46 | 0 |
| 9 | 50 | 8 | M_J50_M8 | 3.27 | 0 |
| 10 | 75 | 4 | M_J75_M4 | 0 | 3.90 |
| 11 | 75 | 6 | M_J75_M6 | 2.70 | 0 |
| 12 | 75 | 8 | M_J75_M8 | 12.67 | 0 |
| 13 | 100 | 4 | M_J100_M4 | 20.56 | 0 |
| 14 | 100 | 6 | M_J100_M6 | 0 | 9.72 |
| 15 | 100 | 8 | M_J100_M8 | 0 | 3.00 |
| 16 | 200 | 10 | L_J200_M10 | 1.79 | 0 |
| 17 | 200 | 20 | L_J200_M20 | 2.88 | 0 |
| 18 | 200 | 30 | L_J200_M30 | 0 | 8.37 |
| 19 | 300 | 10 | L_J300_M10 | 0 | 50.94 |
| 20 | 300 | 20 | L_J300_M20 | 0 | 3.49 |
| 21 | 300 | 30 | L_J300_M30 | 0 | 4.27 |
| 22 | 400 | 10 | L_J400_M10 | 0 | 8.18 |
| 23 | 400 | 20 | L_J400_M20 | 0 | 7.28 |
| 24 | 400 | 30 | L_J400_M30 | 0 | 9.14 |
| Average | | | | 6.95 | 4.51 |

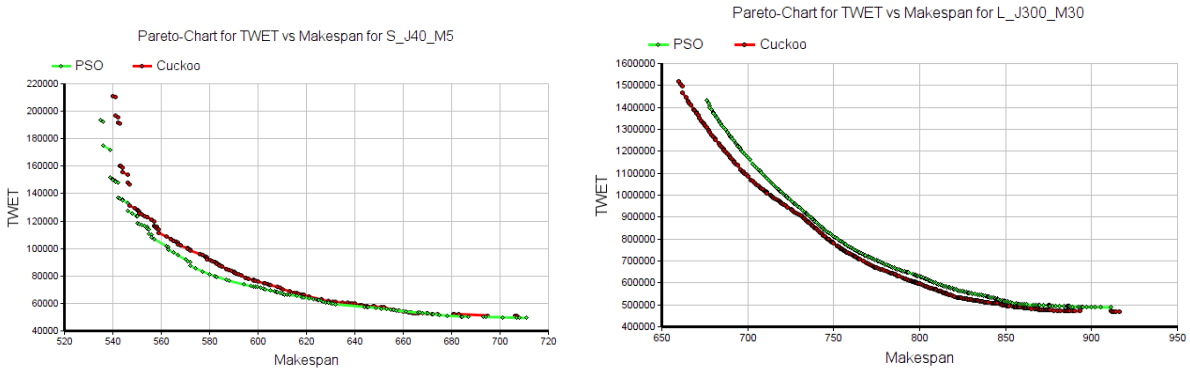


Figure 1. Pareto fronts obtained by the proposed metaheuristics for two problem instances

## 6. Practical implications

The scheduling algorithms presented in this paper find their significance in identifying the processing sequences and schedules that are simultaneously optimal in makespan and TWET objectives for a given set of jobs in unrelated parallel machine scheduling environments operating in a just-in-time setting. The simultaneous optimization of the two objectives is particularly crucial in JIT manufacturing systems striving to minimize the penalty costs due to the early or late completion of jobs, while maintaining efficient utilization of resources and better throughput through continuous production. The scheduling environment characteristics, optimization objectives and problem constraints considered in this paper closely reflect real-world manufacturing setups. However, no work in the literature has presented computationally efficient solution methodologies to solve the problem, which is of NP-hard complexity, for large-sized problem instances that may be encountered in real-world applications. The Pareto-optimal solutions identified using the timing algorithm and the population-based hybrid metaheuristics presented in this paper provide the decision-makers with the necessary insights to choose a solution from the Pareto-optimal solution set, in accordance with their relative priorities for the objectives. Providing the decision makers with such flexibility in choosing from the set of optimal solutions, each corresponding to different trade-offs between the objectives, is crucial in determining the process efficiency and resource utilization of manufacturing industries under fluctuating market demands. Furthermore, the hybrid metaheuristic algorithms presented for the UPMSP in this paper can be easily extended to several real-world manufacturing settings, with diverse scheduling environments such as job shops, flow shops, etc., thereby broadening their practical applicability in the real-world manufacturing contexts.

## 7. Conclusions and future scope

The paper proposes two hybrid metaheuristics, namely the hybrid PAMOCS-LS algorithm and the hybrid MOPSO-LS algorithm, for the simultaneous optimization of makespan and TWET in a JIT-UPMSP. The hybrid metaheuristics generate several job sequences, corresponding to each of which, the proposed exact procedure for trade-off curve generation generates either a piecewise linear convex trade-off curve composed of line segments or a single trade-off point. The Pareto front of multiple trade-off curves, hence obtained, constructed using a method from the literature, is also composed of line segments and points. The comparative performance evaluation of the hybrid metaheuristics shows that the proposed hybrid MOPSO-LS shows superior performance for smaller problem instances, and the proposed hybrid PAMOCS-LS shows superior performance for larger problem instances. This is the first study to present an optimal timing algorithm for the bi-objective optimization of makespan and TWET in a UPMSP, allowing the insertion of idle times in the schedule. This is also the first study to present hybrid population-based metaheuristics for a multi-objective UPMSP in a JIT manufacturing environment, generating a Pareto front composed of line segments and points.

The scheduling complexities that the proposed multi-objective machine scheduling algorithms have addressed not only advance the state of research in scheduling but also open new opportunities for research in diverse scheduling environments across multiple combinations of objectives, thereby contributing towards the operational efficiency of manufacturing systems in JIT environments. The future possibilities of research include suitably adapting other metaheuristic algorithms and comparing their performance with the proposed hybrid population-based metaheuristics, and further studying their performance for different combinations of objectives in the JIT-UPMSP. Another promising research direction is to extend the hybrid population-based metaheuristic algorithms proposed for the UPMSP to diverse scheduling environments such as job shop scheduling, open shop scheduling, flow shop scheduling, etc. Though the proposed hybrid population-based metaheuristics can be easily adapted to diverse machine scheduling environments, every scheduling environment demands developing a unique optimal timing algorithm to generate optimal schedules for a given sequence of jobs between a given set of objectives. This also involves tailoring the operators and techniques within the hybrid population-based meta-heuristics to the specific problem. Additionally, the proposed hybrid population-based metaheuristic algorithms have computational limitations when applied to extremely large-scale problem instances or adaptability limitations when required to dynamically respond to changing problem characteristics, as this research has been restricted to static scheduling environments.

## References

Azevedo, B. F., Montanño-Vega, R., Varela, M. L. R., and Pereira, A., Bio-inspired multi-objective algorithms applied on production scheduling problems, *International Journal of Industrial Engineering Computations*, vol. 14, no. 2, pp. 415–436, 2023.

Babu, S., and Girish, B. S., Pareto-optimal front generation for the bi-objective JIT scheduling problems with a piecewise linear trade-off between objectives, *Operations Research Perspectives*, vol. 12, Article 100299, 2024.

Babu, S., and Girish, B. S., Neighbourhood 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*, vol. 14, Article 100335, 2025.

De CM Nogueira, J.P., Arroyo, J.E.C., Villadiego, H.M.M. and Gonçalves, L.B., Hybrid GRASP heuristics to solve an unrelated parallel machine scheduling problem with earliness and tardiness penalties, *Electronic Notes in Theoretical Computer Science*, vol. 302, pp. 53-72, 2014.

Đurasević, M., and Jakobović, D., Heuristic and Metaheuristic Methods for the Parallel Unrelated Machines Scheduling Problem: A Survey, *Artificial Intelligence Review*, vol. 56, pp. 3181–3289, 2023.

Girish, B. S., Habibullah, H., and Dileeplal, J., Minimizing the total makes for a sequence of operations in job shops, *RAIRO-Oper. Res.*, vol. 56, no. 4. pp. 2621–2649, 2022.

Girish, B. S., and Jawahar, N., A particle swarm optimization algorithm for flexible job shop scheduling problem, *Proceedings of the 2009 IEEE International Conference on Automation Science and Engineering,* pp. 298–303, Bangalore, India, September 9, 2009.

Jacquin, S., Dufossé, F., and Jourdan, L., An exact algorithm for the bi-objective timing problem, *Optimization Letters*, vol. 12, no. 4, pp. 903–914, 2018.

Lee, Y. H., and Pinedo, M., Scheduling jobs on parallel machines with sequence-dependent setup times. *European Journal of Operational Research*, vol. 100, no. 3, pp. 464–474, 1997.

Nartu, T. R., Matta, M. S., Koratana, S., and Bodda, R. K., A fuzzified Pareto multiobjective cuckoo search algorithm for power losses minimization incorporating SVC, *Soft Computing*, vol. 23, no. 21, pp. 10811–10820, 2019.

Neufeld, J. S., Schulz, S., and Buscher, U., A systematic review of multi-objective hybrid flow shop scheduling. *European Journal of Operational Research*, vol. 309, no. 1, pp. 1–23, 2023.

Sterna, M., Late and early work scheduling: A survey. *Omega*, vol. 104, Article 102453, 2021.

Sunday, D., *Distance between Lines and Segments with their Closest Point Approach*, Available: http://geometryalgorithms.com/Archive/algorithm0106/algorithm0106.htm, 2006.

Wang, H., Li, R., and Gong, W., Minimizing tardiness and makespan for distributed heterogeneous unrelated parallel machine scheduling by knowledge and Pareto-based memetic algorithm, *Egyptian Informatics Journal*, vol. 24, no. 3, Article 100383, 2023.

Ying, K.-C., Pourhejazy, P., and Huang, X.-Y., Revisiting the development trajectory of parallel machine scheduling, *Computers & Operations Research*, vol. 168, Article 106709, 2024.

## Biographies

**Sona Babu** is a PhD Scholar in the Department of Aerospace Engineering at the Indian Institute of Space Science and Technology, Thiruvananthapuram, Kerala, India. She has over three years of academic research experience in Industrial Engineering and Operations Research, and more than four years of professional experience as a Software Engineer in the IT industry. She holds a BTech in Electronics and Communication Engineering from the University of Kerala (2018) and an MBA in Operations from The ICFAI University, Tripura (2021). Her research focuses on developing computationally efficient scheduling algorithms for just-in-time manufacturing systems. Her work has been published in top-quartile journals and was recognized with the Third Best Paper Award under the PhD Scholar category at the 2023 ORSI–ICBAI conference jointly organized by IISc Bangalore and IIM Bangalore.

**Dr. B. S. Girish** is working as an Associate Professor in the Department of Aerospace Engineering at the Indian Institute of Space Science and Technology, Thiruvananthapuram, Kerala, India. He holds a BTech degree in Production Engineering from the University of Calicut (2001), ME in Manufacturing Technology from Regional Engineering College Tiruchirapalli (2003), and PhD in Mechanical Engineering from Anna University Chennai (2009). He has over 21 years of academic experience and has published papers in reputed journals and conferences. His research mainly focuses on developing efficient exact and heuristic optimization approaches for various manufacturing systems as well as aerospace systems. He has been a reviewer for several reputed international journals and conferences and received a best reviewer award and a couple of best paper awards in international conferences.