Babu, Sona; Girish, B. S.

Article

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

Provided in Cooperation with:
Elsevier

Suggested Citation: Babu, Sona; Girish, B. S. (2025) : 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, ISSN 2214-7160, Elsevier, Amsterdam, Vol. 14, pp. 1-30, https://doi.org/10.1016/j.orp.2025.100335

This Version is available at:
https://hdl.handle.net/10419/325812

Operations Research Perspectives 14 (2025) 100335

Contents lists available at ScienceDirect

Operations Research Perspectives

journal homepage: www.elsevier.com/locate/orp

Neighbourhood search-based metaheuristics for the bi-objective Pareto optimization of total weighted earliness-tardiness and makespan in a JIT single machine scheduling problem

Sona Babu, B.S. Girish *

Department of Aerospace Engineering, Indian Institute of Space Science and Technology, Valiamala, Thiruvananthapuram, Kerala 695547, India

ARTICLE INFO

Keywords:
Pareto front generation
Variable neighbourhood descent
Total weighted earliness and tardiness
Makespan
Just-in-time manufacturing
Single machine scheduling

ABSTRACT

This paper studies the simultaneous minimization of total weighted earliness-tardiness (TWET) and makespan in a just-in-time single-machine scheduling problem (JIT-SMSP) with sequence-dependent setup times and distinct due windows, allowing idle times in the schedules. Multiple variants of variable neighbourhood descent (VND) based metaheuristic algorithms are proposed to generate Pareto-optimal solutions for this NP-hard problem. An optimal timing algorithm (OTA) is presented that generates a piecewise linear convex trade-off curve between the two objectives for a given sequence of jobs. The trade-off curves corresponding to the sequences of jobs generated in the proposed metaheuristics are trimmed and merged using a Pareto front generation procedure to generate the Pareto-optimal front comprising line segments and points. The computational performance of the proposed VND-based metaheuristic algorithms is compared with state-of-the-art metaheuristic algorithms from the literature on test instances of varying sizes using four performance metrics devised to compare Pareto fronts comprising line segments and points. The performance comparisons reveal that a proposed variant of backtrack-based iterated VND with multiple neighbourhood structures outperforms the other algorithms in most performance metrics.

1. Introduction

Single-machine scheduling problem (SMSP) is one of the most extensively researched scheduling problems in the literature [1]. The problem has widespread applications in the field of manufacturing and computer science in optimizing resource utilization [1]. The problem is also considered a building block in understanding the basic scheduling concepts because it provides a simplified model to investigate various performance measures and solution techniques that can be further extended to other more complex scheduling problems [2,3]. Several problem environments relating to single-machine scheduling have been studied in the literature [4], and several of its variants and extensions have been investigated that consider additional parameters and constraints, viz. sequence-dependent setup times (SDST), machine availability constraints, dynamic job arrivals, stochastic processing times, preemptive jobs, etc. [2,5]. Sequence-dependent setup time, i.e., the time required to switch between jobs on a machine, is one of the most widely considered problem characteristics in machine scheduling problems in the literature [1]. This paper focuses on the basic SMSP with SDST, which involves scheduling n jobs on a single machine where each job requires a non-preemptive single operation to be performed on the continuously available machine. All the jobs are available for processing at time zero, and the job processing times are deterministic.

Several flow-time and due-date-based performance measures have been explored in the SMSP, viz., makespan, total weighted flow time, total weighted tardiness, maximum tardiness, total weighted earliness-tardiness, and total number of tardy jobs [1,4]. These performance measures are formulated as either a single objective function or a multi-objective function to simultaneously optimize a set of objectives. Multi-objective optimization problems can be tackled either by combining the multiple objective functions into a single weighted objective function, assigning weights to the objectives based on their priorities, or by using Pareto-based optimization methods to generate a non-dominated solution set considering all potential trade-offs between the objectives. This paper considers the Pareto-based bi-objective optimization of total weighted earliness-tardiness (TWET) and makespan in the SMSP with SDST.

Minimization of TWET emerged as an important objective with the

* Corresponding author.
E-mail address: girish@iist.ac.in (B.S. Girish).

https://doi.org/10.1016/j.orp.2025.100335
Received 5 December 2024; Received in revised form 24 February 2025; Accepted 20 March 2025
Available online 22 March 2025

advent of just-in-time (JIT) production systems, whereas traditional manufacturing systems focused mainly on the minimization of tardiness [6]. The JIT philosophy strives to achieve zero inventory while satisfying customer demands on time [7]. The expected delivery times for each job in a JIT scheduling problem can either be a due date or a timespan, known as a due window. Several works addressing the bi-objective optimization of earliness and tardiness penalties and other objectives in machine scheduling environments exist in the literature that focus on assigning due windows to the jobs [8–12]. The problem considered in this paper, namely JIT-SMSP, considers due windows for each job, comprising an earliest and a latest due date, beyond which a penalty is imposed [13]. If a job is completed before its earliest due date, it is termed an early job and the losses in the form of inventory holding costs, penalties for early delivery to customers, etc., contribute to the earliness penalty. If the job is completed after its latest due date, it is termed a tardy job, and the loss of customer reputation, the opportunity cost of lost sales, etc., contribute to the tardiness penalty. The total earliness and tardiness due to each factor are, in most cases, quantified as weights and hence are termed weighted earliness and tardiness penalties. Generation of the optimal TWET schedule requires completing the jobs within their respective due windows or as close as possible to their due windows, which may result in idle times in the schedule, particularly when the jobs have distinct due windows [14]. Several real-world JIT scheduling scenarios demanding the minimization of TWET exist in the literature such as in the workload control studies in make-to-stock manufacturing applications, semiconductor wafer fabrication scheduling, steelmaking scheduling, aircraft landing problem, etc. [15–19].

Makespan is another objective intended to be minimized in conjunction with TWET in this paper. Makespan refers to the completion time of the last job in the sequence. Minimizing makespan helps to utilize time and resources efficiently as it generates tighter schedules. The makespan objective does not apply to the basic SMSP, as the completion time of the last job is independent of the sequence and is equal to the sum of the processing times of all the jobs in the problem instance [4]. The makespan objective is relevant only when additional parameters and constraints are considered, such as the sequence-dependent setup times [5], the machine availability constraints [20–22], etc. Since the minimization of TWET results in scheduling the jobs closer to their due windows with inserted idle times [14], the best schedule with respect to the TWET objective need not be the best with respect to the makespan objective and vice versa. This necessitates a trade-off between the two objectives, which is the impetus for this paper. A compromised TWET schedule with a reduced makespan would result in a relatively shortened scheduling window, freeing up the resources to begin the subsequent scheduling window early. This helps ease congestion in successive scheduling windows and improve the overall TWET when increased customer demands are anticipated. The simultaneous minimization of TWET and makespan is significant in real-world JIT scheduling scenarios such as in steelmaking and continuous casting scheduling, surgery scheduling, etc. [23–25]. In such scenarios, a Pareto set of non-dominated solutions considering the two objectives would aid the decision-maker in choosing a suitably compromised schedule.

The SMSP with the minimization of TWET and makespan as single objectives is well known to be strongly NP-hard [26,27]. Simultaneously optimizing the two objectives would make the problem even more complex to solve. Much of the research on these objectives has focused on developing heuristic and metaheuristic algorithms. Therefore, this paper proposes Pareto-based metaheuristic algorithms for the bi-objective optimization of TWET and makespan in the SMSP with SDST and distinct due windows.

The remaining sections of the paper are organized as follows. Section 2 presents the literature review, Section 3 presents the problem formulation, Section 4 presents the solution methodologies, Section 5 presents the computational study of the solution methodologies, and Section 6 concludes with the scope for future work.

2. Literature review

Single objective optimization of TWET and makespan has been extensively researched in different scheduling environments in the literature [1,5,28]. Numerous works also exist that consider the optimization of each of these objectives in conjunction with other objectives, viz. total flow time, total earliness, total tardiness, number of tardy jobs, maximum earliness, maximum tardiness, compression and expansion cost of processing times, work-in-process inventory costs, etc. [13, 29–32]. However, relatively little research exists on the simultaneous minimization of the TWET and makespan objectives.

Some of the existing works on the multi-objective optimization of TWET and makespan have considered simplifying the concurrent bi-objective optimization problem into a weighted single objective optimization problem by assigning weights to the objectives based on their corresponding priorities [33–35]. Very few works exist on the simultaneous minimization of the bi-objectives using Pareto-based optimization approaches, particularly in the SMSP environment. Table 1 presents a summary of the literature review on the Pareto-based optimization of TWET and makespan in the SMSP and several other machine scheduling environments. Since this paper studies the basic SMSP with sequence-dependent setup times (SDST), the review of literature has been restricted to static scheduling environments with deterministic processing times.

Gao et al. [36] proposed a parallel genetic algorithm based on a vector group encoding method and an immune method for the Pareto-based optimization of TWET and makespan in a non-identical parallel machine scheduling problem (PMSP) subjected to a special process constraint. Gao [37] presented a vector artificial immune system algorithm for the Pareto-based minimization of the two objectives in a non-identical PMSP subjected to a special process constraint. Fakhrzad et al. [38] proposed a hybrid genetic algorithm to minimize TWET and makespan simultaneously in a job shop scheduling problem considering SDST. Tajbakhsh et al. [39] proposed a hybrid particle swarm optimization-genetic algorithm for the Pareto-based bi-objective optimization of TWET and makespan in a three-stage manufacturing system, including the machining, assembly and batch processing stages. Abedi et al. [40] considered the simultaneous minimization of the two objectives in identical parallel batch processing machines operating in a JIT environment with arbitrary job sizes, unequal job release times and capacity limits. They presented a non-dominated sorting genetic algorithm II (NSGA-II) and a multi-objective imperialist competitive algorithm to solve the problem. Zade et al. [41] proposed a Pareto-based multi-objective particle swarm optimization algorithm for the bi-objective optimization of TWET and makespan in the SMSP with periodic preventive maintenance. Zarandi and Kayvanfar [42] considered the simultaneous minimization of TWET and makespan, along with the compression and expansion costs of processing times in an identical PMSP. They implemented two multi-objective evolutionary algorithms, namely non-dominated sorting genetic algorithm II (NSGA-II) and non-dominated ranking genetic algorithm (NRGA), to generate the Pareto-optimal front. Rad et al. [43] applied an ε-constraint method to validate their proposed model for the simultaneous minimization of TWET and makespan in a two-stage assembly flow shop scheduling problem. Shahidi-Zadeh et al. [44] considered the simultaneous minimization of makespan, TWET and job incompatibility in a batch PMSP and presented a mathematical model that was solved using the ε-constraint method. Shahriari et al. [45] considered the simultaneous minimization of the two objectives, TWET and Makespan, in the SMSP, considering periodic preventive maintenance and restricting the maximum number of jobs allowed in a certain period. They proposed a bi-objective mixed integer model and implemented multi-objective particle swarm optimization to solve the problem. Xu et al. [46] proposed a multi-objective artificial bee colony algorithm for the

Table 1
Summary of works on the Pareto optimization involving TWET and makespan objectives in SMSP and other scheduling environments.

Author	Problem characteristics	Objectives used	Methodology	Due Date (DD)/ Due Window (DW)	SDST
Gao et al. (2009)	Non-identical PMSP subjected to special process constraint	Makespan vs. TWET	Parallel genetic algorithm (PIGA) based on the vector group encoding and the immune method	DW	
Gao (2010)	Non-identical PMSP subjected to special process constraint	Makespan vs. TWET	Vector artificial immune system (VAIS) algorithm	DW	
Fakhrzad et al. (2013)	Job shop scheduling with SDST	Makespan vs. TWET	Hybrid genetic algorithm (GA)	DW	√
Tajbakhsh et al. (2014)	Three-stage manufacturing system with machining, assembly and batch processing stages	Makespan vs. TWET	Hybrid particle swarm optimization – genetic algorithm (PSO-GA)	DD	
Abedi et al. (2015)	Identical parallel batch processing machines with arbitrary job sizes, unequal job release times and capacity limits	Makespan vs. TWET	Non-dominated sorting genetic algorithm II (NSGA-II) and multi-objective imperialist competitive algorithm (MOICA)	DD	
Zade et al. (2015)	SMSP with periodic preventive maintenance	Makespan vs. TWET	Multi-objective particle swarm optimization (MOPSO) algorithm	DD	
Zarandi and Kayvanfar (2015)	Identical PMSP with controllable processing times	Makespan vs. Sum of TWET and compression and expansion costs of processing times	Non-dominated sorting genetic algorithm II (NSGA-II) and non-dominated ranking genetic algorithm (NRGA)	DD	
Rad et al. (2015)	Two-stage assembly flow shop scheduling problem	Makespan vs. TWET	ε-constraint method	DD	
Shahidi-Zadeh et al. (2015)	Batch PMSP with maximum allowable job incompatibility	Tri-objective optimization of Makespan, TWET and Incompatibility of jobs in batches	ε-constraint method	DD	
Shahriari et al. (2016)	SMSP considering periodic preventive maintenance and restricting the maximum number of jobs allowed in a certain period	Makespan vs. TWET	MOPSO	DD	
Xu et al. (2016)	Hybrid flow shop scheduling problem with unrelated parallel machines	Tri-objective optimization of Makespan, TWET and Total waiting time	Multi-objective artificial bee colony (ABC) algorithm	DD	√
Shahvari and Logendran (2017)	Batch processing problem with dual resources on unrelated parallel machines	Makespan vs. Production cost including TWET	Particle swarm optimization (PSO) - based search algorithms	DD	√
Shahidi-Zadeh et al. (2017)	Unrelated parallel batch processing scheduling problem with job release times and batch capacity constraints	Makespan vs. Sum of TWET and machine purchasing costs	Multi-objective harmony search (MOHS) algorithm	DD	
Shen (2019)	Uncertain uniform PMSP with job deterioration and learning effect with uncertainties in job processing times, due dates, deterioration rates and learning rates	Makespan vs. TWET	Hybrid algorithm with mixed dispatching rules	DD	
Jia et al. (2020)	Parallel batch scheduling problem	Tri-objective optimization of Makespan, TWET and total energy consumption	History-guided multi-objective evolutionary algorithm based on decomposition	DD	
Shao et al. (2021)	Distributed hybrid flow shop scheduling problem	Tri-objective optimization of Makespan, TWET and Total workload	Multi-objective evolutionary algorithm based on multiple neighbourhood local search (MOEA-LS)	DD	
Wei et al. (2021)	Energy-efficient job shop scheduling problem	Tri-objective optimization of Makespan, TWET and Non-processing energy consumption	Unified non-dominated sorting genetic algorithm-III (U-NSGA-III)	DD	
Ampry et al. (2022)	Unrelated parallel batch scheduling problem	Makespan vs. Sum of TWET and Machine purchasing costs	Multi-objective harmony search algorithm	DD	
This paper	SMSP	Makespan vs. TWET	Variable neighbourhood descent (VND)-based algorithms	DW	√

Pareto-based optimization of makespan, TWET and total waiting time in a hybrid flow shop scheduling problem containing unrelated parallel machines. Shahvari and Logendran [47] proposed particle swarm optimization-based search algorithms for the simultaneous minimization of makespan in conjunction with the production cost, including TWET in a batch processing problem with dual resources on unrelated parallel machines. Shahidi-Zadeh et al. [48] proposed a multi-objective harmony search algorithm to simultaneously minimize the makespan, TWET and the purchasing cost of machines in an unrelated parallel batch processing scheduling problem considering job release times and batch capacity constraints. Shen [49] considered an uncertain uniform PMSP with job deterioration and a learning effect with uncertainties in job processing times, due dates, deterioration rates and learning rates. They proposed a hybrid algorithm based on dispatching rules for the Pareto-based bi-objective minimization of TWET and makespan. Jia et al. [50] proposed an evolutionary algorithm based on decomposition to simultaneously minimize the makespan, TWET and total energy consumption in a parallel batch scheduling problem. Shao et al. [51] proposed an evolutionary algorithm employing local search with multiple neighbourhoods to minimize TWET, makespan and total workload in a distributed hybrid flow shop scheduling problem. Wei et al. [52] proposed a multi-objective genetic algorithm for the simultaneous minimization of TWET, makespan and non-processing energy consumption in an energy-efficient job shop scheduling problem. Ampry et al. [53] presented a multi-objective harmony search algorithm to simultaneously minimize the makespan, TWET and the cost of purchasing machines in an unrelated parallel batch processing scheduling problem.

From the above literature review, it is evident that not much research exists in the literature on the Pareto-based optimization of the two

objectives, TWET and makespan, in the SMSP, though Pareto-based optimization on other combinations of objectives in the SMSP has been actively researched [13,54–61]. The extent of research on different machine scheduling problems reveals the importance of generating the trade-off between the two objectives, which is evident from the above literature review. The above literature review also reveals that the sequence-dependent setup times and due windows have not been considered in most of the works and heuristic and metaheuristic approaches have been the most studied solution methodologies, particularly for solving larger-sized instances with 100 or more jobs. Though most of the existing works on multi-objective optimization considering TWET as one of the objectives considered the TWET objective from a JIT perspective, they did not account for idle times on the machines while generating schedules, i.e., all the jobs are scheduled at their earliest possible start times rather than closer to their respective due dates. Therefore, the schedule generated for a given sequence of jobs will always result in a single point on the Pareto chart. In other words, for a given sequence of jobs, only one optimal schedule exists with its optimal TWET and the corresponding makespan value. Arroyo et al. [13] considered Pareto-based multi-objective optimization of TWET and total flow time (TFT) in the SMSP, allowing idle times to be inserted into the schedule. However, they considered the two objectives as a lexicographic function with TWET as the primary objective and TFT as the secondary objective, which results in a single trade-off point on the Pareto chart corresponding to a given sequence of jobs.

Jacquin et al. [62] considered the problem of simultaneously minimizing total earliness (TE) and total tardiness (TT) in the SMSP, allowing idle times to be inserted into the schedules. They showed that each sequence of jobs results in a piecewise linear convex trade-off curve between the two objectives. Babu and Girish [63] considered the problem of Pareto-based bi-objective optimization of TWET and total flowtime (TFT) in the SMSP with idle times allowed to be inserted in the schedules. They presented an optimal timing algorithm to generate optimal schedules corresponding to a given sequence of jobs and showed that the trade-off relationship between the two objectives is a piecewise linear convex trade-off curve when idle times exist in the schedules. The bi-objective optimization of TWET and makespan in the SMSP considered in this paper also allows idle times to be inserted in the schedules to determine the Pareto-optimal solutions. Therefore, each sequence of jobs may result in a piecewise linear trade-off curve, as in the above-mentioned cases involving the bi-objectives of TE-TT and TWET-TFT in the SMSP. This will require devising an optimal timing algorithm (OTA) or utilising an optimization solver to generate schedules corresponding to a given sequence of jobs. We adopt an OTA presented in the literature [64,65] and extend it to generate optimal schedules corresponding to the job sequences generated by the proposed Pareto-based metaheuristic algorithms. We also adopt an exact method of Pareto front generation proposed by Babu and Girish [63] to generate the Pareto-optimal front from multiple sequences of jobs, where each sequence of jobs has an associated piecewise linear trade-off curve comprising line segments on the Pareto chart. Babu and Girish [63], in their work, employed a greedy local search heuristic with a pairwise interchange neighbourhood generation mechanism and showed that their proposed method of Pareto-optimal front generation is computationally efficient than an upper envelop algorithm adopted from the literature and is suitable for implementation within metaheuristic algorithms. We adopted their Pareto-optimal front generation method and devised computationally efficient Pareto-based metaheuristic algorithms to solve the NP-hard problem under consideration. The Pareto-based metaheuristic algorithms presented in this paper are based on the variable neighbourhood descent (VND) approach, which is a well-known neighbourhood search-based metaheuristic algorithm for solving single objective and multi-objective combinatorial optimization problems. The proposed VND-based approaches have been derived by improving the VND-based methods existing in the literature, in terms of computational efficiency, by applying various improvement mechanisms of neighbourhood generation and perturbation. This paper also suitably adapts the existing bi-objective-based performance metrics to compare the proposed metaheuristic algorithms since the Pareto-optimal front comprises both points and line segments, while most of the existing performance metrics are designed only for points on the Pareto chart. The performance of the proposed VND-based metaheuristics have been compared with other state-of-the-art neighbourhood search-based metaheuristic algorithms and a population-based metaheuristic adopted from the literature.

3. Problem formulation

The bi-objective SMSP discussed in this paper is as follows. Let n denote the number of jobs that are to be processed on a single machine. Let i denote the job index, and j denote the position index of the jobs in the sequence. Let P_i denote the processing time of job i and $S_{i'i}$ denote the setup time to switch from job i' to job i. Let $[de_i, dt_i]$ denote the due window of job i, where de_i and dt_i denote the earliest and the latest allowable due dates of job i, respectively. Let C_i denote the completion time scheduled for job i. Then, the earliness is defined as $E_i = max(0, de_i - C_i)$, and the tardiness is defined as $T_i = max(0, C_i - dt_i)$. Let α_i and β_i, respectively, be the weights associated with the early and tardy completion of a job i. The assumptions and notations used in the problem formulation are listed below.

Assumptions:

- The single machine is continuously available.
- All the jobs are available at time zero.
- Each job i requires a single operation to be performed on the machine.
- The machine can process only one job at a time.
- The job descriptors are deterministic and known beforehand.
- No setup time is required for the job assigned to the first position in the sequence.
- Job preemptions are not allowed.

List of notations used in the proposed model:

n	: Number of jobs
i	: Job index (i=1,2,…,n)
j	: Position index of jobs (j=1,2,…,n)
P_i	: Processing time of job i
$S_{i'i}$	: Setup time to switch from job i' to job i
de_i	: Earliest allowable due date of job i
dt_i	: Latest allowable due date of job i
α_i	: Earliness penalty of job i
β_i	: Tardiness penalty of job i

The mathematical formulation [13] is as follows.

Decision Variables:

$$x_{ij} = \begin{cases} 1: \textit{ if job } i \textit{ is assigned to position } j \textit{ in the sequence} \\ 0: \textit{otherwise} \end{cases}$$

C_i = completion time of job i

E_i = earliness of job i

T_i = tardiness of job i

Objective:

$$\text{Minimize } \sum_{i=1}^{n} (\alpha_i E_i + \beta_i T_i) \quad (1)$$

$$\text{Minimize } max_i(C_i) \quad (2)$$

Subject to:

$$\sum_{i=1}^{n} x_{ij} = 1 \ \forall j \tag{3}$$

$$\sum_{j=1}^{n} x_{ij} = 1 \ \forall i \tag{4}$$

$$C_i \geq P_i \ \forall i \tag{5}$$

$$C_i \geq C_{i'} + P_i + S_{i'i} - H(1 - x_{i'j}) - H(1 - x_{ij+1}) \forall i, i', j : i \neq i' \ \& j = 1, 2, \ldots, n-1 \tag{6}$$

$$T_i \geq C_i - dt_i \ \forall i \tag{7}$$

$$E_i \geq de_i - C_i \ \forall i \tag{8}$$

$$x_{ij} \in \{0, 1\} \ \forall i, j \tag{9}$$

$$C_i \geq 0, \ E_i \geq 0, \ T_i \geq 0 \ \forall i \tag{10}$$

In the mathematical formulation, x_{ij}, C_i, E_i and T_i are the decision variables, and solving the problem aims to obtain the optimal values of these variables. The objective functions (1) and (2) express the weighted sum of earliness and tardiness costs and the makespan, respectively. Constraints (3) and (4) ensure that each position in the sequence is allocated to only one job, and each job is allocated to only one position in the sequence. Constraint (5) ensures that the completion time of the first job in the sequence is not less than its processing time. Constraint (6) is a disjunctive constraint that is active for all consecutive pairs of jobs in the sequence and non-active for all non-consecutive pairs of jobs. H denotes a large positive integer. Constraints (5) and (6) together ensure the generation of a feasible completion time for each job based on its position in the sequence. Constraints (7) and (8) associate the tardiness and earliness of each job with its completion time and its latest and earliest due dates, respectively. Constraints (9) and (10) define the variable bounds.

Algorithm 1
Generation of TWET-Makespan trade-off curve.

Data: $n, \sigma, P_i, de_i, dt_i, \ \alpha_i, \ \beta_i, \forall i \in \sigma, \ S_{ii'} \ \forall \ i, i' : \ i' \ \in \ \sigma, i \ \in \ \sigma$
for $(k \ = \ 1 \ to \ n)$ **do**
  if $(k = 1)$ **then**
    $a \leftarrow \sigma[k]$
    $C_a \ \leftarrow de_a$
    $B \leftarrow \ \{k\}$
  else
    $a \leftarrow \sigma[k-1]$
    $b \leftarrow \sigma[k]$
    if $(C_a + P_b + \ S_{ab}) < de_b$ **then**
      $C_b \leftarrow de_b$
      $B \leftarrow \emptyset$
      $B \leftarrow \{k\}$
      if $(k \ = \ n)$ **then**
        $t \ \leftarrow \ 1$
        $SAVE\_BREAK\_POINT(t)$
        $LEFT\_SHIFT(k)$
      end
    else
      $C_b \ \leftarrow C_a + P_b + \ S_{ab}$
      if $(k \ = \ n)$ **then**
        $t \ \leftarrow \ 1$
        $SAVE\_BREAK\_POINT(t)$
      end
      $B \leftarrow B \ \cup \ \{k\}$
      $LEFT\_SHIFT(k)$
    end
  end
end
Function $SAVE\_BREAK\_POINT(t)$
  $G_t \leftarrow \sum_{i=1}^{n} (\alpha_i \ max(0, de_i \ - \ C_i) \ + \ \beta_i \ max(0, C_i \ - \ dt_i))$
  $M_t \leftarrow \ \max_{1 \leq i \leq n} \{C_i\}$
end

4. Solution methodologies

This section first presents the solution representation for the SMSP with SDST considered in this paper and the procedure for the generation of the TWET-makespan trade-off curve for a given solution. Further, the procedure adopted from the literature for the generation of a Pareto-optimal front from multiple TWET-makespan trade-off curves is described. Subsequently, the procedure for the generation of initial solutions and the neighbourhood search-based multi-objective metaheuristic optimization algorithms are presented.

4.1. Solution representation and generation of the TWET-makespan trade-off curve

The solution representation used in the proposed metaheuristics is a permutation of the job indexes representing the order in which the jobs are sequenced for processing on the machine. In the SMSP, the jobs can be sequenced in $n!$ possible ways. Let σ represent an ordered set comprising a sequence of n jobs to be processed on a single machine. The TWET-makespan trade-off curve for a given sequence of jobs is generated using the optimal timing algorithm (OTA) presented in Algorithms 1 and 2. The procedure first generates the optimal TWET schedule and its corresponding optimal makespan. Subsequently, it iteratively reduces the makespan and generates the breakpoints to obtain the optimal piecewise linear TWET-makespan trade-off curve. The procedure to generate the optimal TWET schedule corresponding to a given sequence of jobs shown in Algorithms 1 and 2 is based on the timing algorithms proposed in the literature [14,64,65]. However, the timing algorithms presented in the literature for the SMSP consider a single due date corresponding to each job. The procedure shown in Algorithms 1 and 2 to generate the TWET-makespan trade-off curve is described as follows.

In Algorithm 1, a and b represent two consecutive jobs in σ, C_a and C_b are the respective completion times of jobs a and b, S_{ab} is the setup time to switch to job b from job a on the machine, and P_a and P_b are the processing times of jobs a and b, respectively. Let σ_i represent an ordered set comprising the partial sequence of the first i number of jobs in σ $\forall i \in \{1, 2, \ldots, n\}$ i.e., $\sigma_i \subseteq \sigma$, and $\sigma[j]$ represent the job identifier at the j^{th} position in the sequence. The first job a in σ is initially scheduled to be completed at its earliest due date (i.e. de_a), as shown in step 4 of Algorithm 1. The TWET associated with this assignment in the partial sequence σ_1 will be 0. The subsequent jobs in σ are scheduled successively as close as possible to their earliest due dates, and the TWET associated with the jobs in the respective partial sequence is optimized following the addition of each job. Scheduling the jobs close to their earliest due dates implies that the completion times of the jobs are as close as possible to their respective earliest due dates. If the earliest due date of job b (i.e. de_b) is less than $C_a + P_b + S_{ab}$, then job b is scheduled to be completed exactly at de_b, as shown in step 10 of Algorithm 1. This implies that there will be an idle time between the jobs a and b, and the TWET associated with this partial sequence and its corresponding schedule will be optimal. On the other hand, if $C_a + P_b + S_{ab}$ exceeds de_b, then the job is assigned its completion time (i.e. C_b) as equal to $C_a + P_b + S_{ab}$, as shown in step 19 of Algorithm 1. This implies that no idle time exists between the completion times of jobs a and b, i.e. the jobs are contiguous with each other. In other words, the completion times of jobs a and b are clustered around their earliest due dates. The TWET associated with this partial schedule need not be optimal, which needs to be further optimized by left shifting the last job assigned to the partial sequence by invoking the function *LEFT_SHIFT* in step 25 of Algorithm 1.

Algorithm 2 presents the left shifting procedure (*LEFT_SHIFT*), which left shifts the last job assigned at position k in σ_k (i.e. the job $\sigma[k]$) along with the set of preceding contiguously scheduled jobs to optimize the partial schedule without violating the separation constraints between the completion times of consecutive jobs. This set of contiguously scheduled jobs is also called a block and is generated as follows.

Let B represent the block that contains the position identifier k of the last job b in the partial sequence σ_k and the position identifiers of all its preceding contiguous jobs that will allow for a feasible left shifting of the last job b. A feasible left shifting indicates that the last job b in the partial sequence can be left shifted by at least one unit of time without violating the separation constraints between the completion times of jobs given by Eqs. (5) and (6), discussed in Section 3. B is initially assigned the position identifier $k = 1$ of the job in the first position in σ_k, as shown in step 5 of Algorithm 1. Subsequently, if any job b at position k is found to be contiguous with its immediately preceding job at position $k-1$ in σ_k, then the position identifier of job b is added to B, as shown in step 24 of Algorithm 1. However, if a job b at position k in σ_k is non-contiguous with its immediately preceding job at position $k-1$, then the block B is reset to include only the position identifier k, as shown in steps 11 and 12 of Algorithm 1. The TWET cost function corresponding to the jobs with its position identifiers in B, represented by $TWET(B)$, will always be a piecewise linear convex cost function with a minimum point, and left shifting the jobs with position identifiers in B to this minimum point optimizes the TWET cost function corresponding to the partial sequence σ_k (i.e. $TWET(\sigma_k)$) [64,65]. The following theorem explains the optimality of the procedure described in Algorithms 1 and 2 to obtain the minimum $TWET(\sigma)$.

Theorem 1. *The set of jobs with position identifiers in B, which contains the job at the last position k in the partial sequence σ_k as well as all its preceding contiguous set of jobs, when simultaneously left shifted to the minimum point of its cost function $TWET(B)$, optimizes $TWET(\sigma_k)$.*

Proof. For a given partial sequence σ_k, the TWET cost function of the jobs with position identifiers belonging to B for a given partial schedule $S_k = \{C_{\sigma[1]}, C_{\sigma[2]}, \ldots, C_{\sigma[k]}\}$ can be expressed as

$$TWET(B) = \sum_{j \in B} \left(\alpha_{\sigma[j]}\, max\left(0,\ de_{\sigma[j]} - C_{\sigma[j]}\right) + \beta_{\sigma[j]}\, max\left(0,\ C_{\sigma[j]} - dt_{\sigma[j]}\right) \right) \tag{11}$$

For a specific S_k, a few jobs with position identifiers in B will be early from their respective earliest due dates (i.e. $C_{\sigma[j]} \le de_{\sigma[j]}$), and a few other jobs will be tardy from their respective latest due dates (i.e. $C_{\sigma[j]} \ge dt_{\sigma[j]}$). The remaining jobs with position identifiers in B will be scheduled within their respective due windows (i.e. $de_{\sigma[j]} \le C_{\sigma[j]} \le dt_{\sigma[j]}$) and do not contribute to $TWET(B)$. Let $EY \in B$ be the set of position identifiers corresponding to the early jobs, $TY \in B$ be the set of position identifiers corresponding to the tardy jobs, and $DW \in B$ be the set of position identifiers of jobs which are scheduled within their respective due windows in B. Then, the TWET cost function for a given S_k can be rewritten as [63]

$$TWET(B) = \sum_{j \in EY} \alpha_{\sigma[j]} \left(de_{\sigma[j]} - C_{\sigma[j]} \right) + \sum_{j \in TY} \beta_{\sigma[j]} \left(C_{\sigma[j]} - dt_{\sigma[j]} \right) \tag{12}$$

For a specific value of $C_{\sigma[k]}$, where $\sigma[k]$ is the job at the last position in σ_k, the TWET cost function can be rewritten as

$$\begin{aligned} TWET(B) = {} & \left(\sum_{j \in TY} \beta_{\sigma[j]} - \sum_{j \in EY} \alpha_{\sigma[j]} \right) C_{\sigma[k]} + \sum_{j \in EY} \alpha_{\sigma[j]} \left(de_{\sigma[j]} + T_{\sigma[j]} \right) \\ & - \sum_{j \in TY} \beta_{\sigma[j]} \left(dt_{\sigma[j]} + T_{\sigma[j]} \right) \end{aligned} \tag{13}$$

where $T_{\sigma[j]}$ denotes the time gap between the completion times of jobs $\sigma[k]$ and $\sigma[j]$, i.e. $T_{\sigma[j]} = C_{\sigma[k]} - C_{\sigma[j]}$, which remains constant for all the jobs in B, when $\sigma[k]$ is varied by left shifting all the jobs with their position identifiers in B by the same amount of time. Therefore, Eq. (13) will be a straight-line equation with a slope

$$SL = \sum_{j \in TY} \beta_{\sigma[j]} - \sum_{j \in EY} \alpha_{\sigma[j]} \tag{14}$$

Algorithm 2
Left shifting procedure.

Function $LEFT\_SHIFT(k)$
  $t_1 \leftarrow 0\ ,\ t_2 \leftarrow 0\ ,\ t_3 \leftarrow 0,\ \delta \leftarrow M\ ,\ l \leftarrow |B|, m \leftarrow min(B)$ ▷ M is a large positive integer
  if $(m = 1)$ **then**
    $\delta \leftarrow C_{\sigma[1]} - P_{\sigma[1]}$
  end
  if $(\delta\ >\ 0)$ **then**
    $SL \leftarrow 0$
    for $(\forall j\ \in B)$ **do**
      $a \leftarrow \sigma[j]$
      if $(C_a \leq\ dt_a)$ **then**
        $SL \leftarrow SL - \alpha_a$
      else if $C_a >\ dt_a$ **then**
        $SL\ \leftarrow SL\ + \beta_a$
      end
    end
    if $(SL\ >\ 0\ or\ k\ =\ n)$ **then**
      if $(l\ <\ k)$ **then**
        $a \leftarrow\ \sigma[m-1], b \leftarrow \sigma[m]$
        $t_1 \leftarrow C_b - P_b - S_{ab} - C_a$
        $\delta\ \leftarrow min(\delta, t_1)$
      end
      $t_2 \leftarrow \underset{j \in B}{min}(C_{\sigma[j]}\ -\ dt_{\sigma[j]}\ :\ C_{\sigma[j]}\ >\ dt_{\sigma[j]})$
      $t_3 \leftarrow \underset{j \in B}{min}(C_{\sigma[j]}\ -\ de_{\sigma[j]}\ :\ de_{\sigma[j]} < C_{\sigma[j]} \leq\ dt_{\sigma[j]})$
      $\delta \leftarrow min(\delta, t_2, t_3 : t_2 > 0,\ t_3 > 0)$
      $C_{\sigma[j]} \leftarrow C_{\sigma[j]} - \delta\ \ \forall j \in B$
      if $(k\ =\ n\ and\ SL\ \geq\ 0)$ **then**
        $t \leftarrow 1$
        $SAVE\_BREAK\_POINT(t)$
      else if $(k\ =\ n\ and\ SL\ <\ 0)$ **then**
        $t \leftarrow t\ +\ 1$
        $SAVE\_BREAK\_POINT(t)$
      end
      if $(\delta\ = t_1)$ **then**
        for $(j\ = k - l\ \ to\ 1)$ **do**
          $a \leftarrow \sigma[j]$
          $b \leftarrow \sigma[j+1]$
          if $(C_b = C_a + P_b + S_{ab})$ **then**
            $B \leftarrow B\ \cup\ \{j\}$
          else
            break
          end
        end
      end
      go to step 2
    end
  end
end

When the set of jobs with position identifiers in B are simultaneously left shifted, the tardy jobs in the set TY get shifted to DW at their respective latest due dates, and the jobs in the set DW get shifted to EY at their respective earliest due dates. This results in the slope SL of the cost function $TWET(B)$ to monotonically decrease, forming a piecewise linear convex curve with breakpoints on the TWET-$C_{\sigma[k]}$ plot with a minimum point, as shown in Fig. 1. Each breakpoint on the TWET-$C_{\sigma[k]}$ plot indicates a job shifting from set TY to DW or from set DW to EY, resulting in a decrease in slope SL. At the minimum point of the cost function $TWET(B)$, its slope SL changes from a non-negative value to a negative value. Shifting the jobs with position identifiers in B to the minimum point on the TWET-$C_{\sigma[k]}$ plot minimizes $TWET(B)$ as well as $TWET(\sigma_k)$, since the jobs belonging to σ_k whose position identifiers are not in B remain unchanged during the left shifting of B.

Since each job in σ_k was successively assigned completion times and their partial schedules optimized by shifting the jobs with position identifiers in B to the minimum point of its TWET cost function as described in Algorithm 1, any job or a set of jobs in σ_k, which is non-contiguous with the jobs with position identifiers in B, will not improve TWET if left shifted along with B. This is because the jobs or the set of contiguous jobs preceding the jobs with position identifiers in B were already optimized to the minimum point of their respective cost functions successively, i.e. in the order TWET(σ_1), TWET(σ_2), …, TWET (σ_{k-1}). Hence, the partial schedule S_k, which is optimized by left shifting B to the minimum point of its cost function, optimizes $TWET(\sigma_k)$.

However, in the process of left shifting of jobs with position identifiers in B to the minimum point of $TWET(B)$ as described above, a job with its position identifier in B can become contiguous with a preceding job whose position identifier is not included in B. In that case, the set B adds to it the position identifiers of the preceding contiguous jobs for a feasible left shifting and left shifts the jobs to the minimum point of the cost function $TWET(B)$ corresponding to the updated set B. Every time set B is updated with position identifiers of the preceding contiguous jobs during left shifting, the slope SL decreases, resulting in a breakpoint on the TWET-$C_{\sigma[k]}$ plot.

From the above theorem, it can be inferred that for a positive value of SL, left shifting of the jobs with position identifiers included in B results in the minimization of $TWET(\sigma_k)$ of the partial sequence σ_k. Therefore, the jobs with position identifiers belonging to B in the partial sequence σ_k are left shifted until the corresponding SL becomes negative or until there is no idle time preceding the job in the first position in σ_k, whichever is encountered first.

In Algorithm 2, δ represents the maximum time units by which the jobs with position identifiers in B can be left shifted without encountering any slope changes due to a job's completion time crossing its latest due date or the earliest due date or the set of jobs whose position identifiers belonging to B becoming contiguous to a preceding job, or no gap is left preceding the first job in σ_k for further left shifting. If the position identifier of the job in the first position in σ_k (i.e. $\sigma[1]$) belongs to B, then δ is updated as the maximum time units by which the first job can be left shifted, as shown in step 4 of Algorithm 2. If the position identifier of the job $\sigma[1]$ does not belong to B, then the maximum time units by which the jobs in block B can be left shifted until it becomes contiguous with its preceding job is determined and assigned to t_1, as shown in step 19 of Algorithm 2. t_2 is the time units by which the block can be left shifted, such that the completion time of a job with its position identifier in B is the first to reach its latest due date, as shown in step 22 of Algorithm 2. t_3 is the time units by which the block can be left shifted, such that a job with its position identifier in B is the first to reach its earliest due date, as shown in step 23 of Algorithm 2. δ is the minimum of t_1, t_2 and t_3 and is chosen as the time units for left shifting the jobs whose position identifiers in σ_k belong to B, as shown in steps 20–24 of Algorithm 2. Due to left shifting, if the jobs with position identifiers belonging to B become contiguous with its preceding job, the position identifiers of the preceding job or the set of preceding contiguous jobs in σ_k are added to the block B, as shown in steps 33–43 of Algorithm 2. The left shifting of the jobs with position identifiers in B proceeds until the corresponding SL becomes negative or no further left shifting is possible. Once all the jobs in σ are successively scheduled in this manner, the optimized TWET G_t and the corresponding makespan M_t for the identified optimal TWET schedule are evaluated. This is implemented in the *SAVE_BREAK_POINT* function as shown in steps 15, 22 and 29–32 of Algorithm 1 and steps 26–28 of Algorithm 2.

The solution generated using the above procedure is a single optimal trade-off point representing the optimal TWET and its corresponding makespan on the Pareto chart representing the objectives space. To generate all possible trade-offs between TWET and makespan on the Pareto chart, the jobs in the optimal TWET schedule are further left shifted to reduce the makespan. The last job in the sequence σ and all its preceding contiguous jobs are continued to be left shifted even though the corresponding SL is negative, as shown by the conditions in step 16 of Algorithm 2. Makespan is hence minimized to the maximum possible extent, resulting in the elimination of idle time from the schedule. The trade-off points corresponding to each value of makespan and the corresponding TWET are stored whenever a change in SL is encountered, as

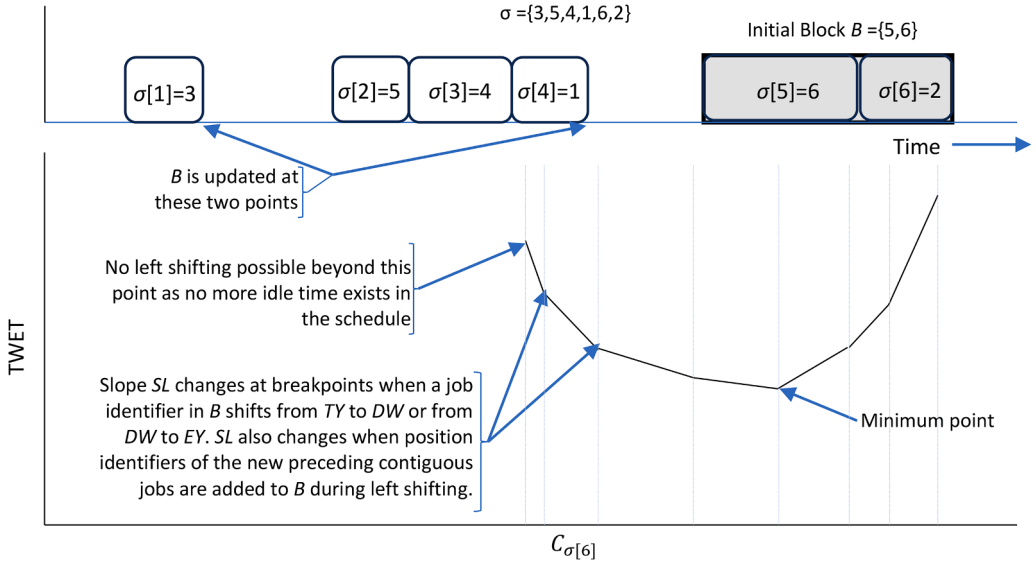


Fig. 1. A typical TWET-$C_{\sigma[k]}$ trade-off plot.

shown in steps 26–32 of Algorithm 2. The breakpoints generated are then connected by line segments, which constitute the optimal TWET-makespan trade-off curve for that particular sequence of jobs. Since the *SL* value monotonically reduces with the left shifting of jobs in the sequence σ, the TWET-makespan relationship will be a piecewise linear convex trade-off curve [14,63–65]. Each point on the TWET-makespan trade-off curve represents a non-dominated solution for the given sequence of jobs. However, if no idle time exists in the optimal TWET schedule, the trade-off relationship between the bi-objectives will be a single point. A numerical illustration of the TWET-makespan trade-off curve generation procedure shown in Algorithms 1 and 2 is presented in Appendix A. The following theorem explains the optimality of the procedure described above to obtain the optimal TWET-makespan trade-off curve.

Theorem 2. *The set of jobs with its position identifiers in B in the optimal TWET schedule, when simultaneously left shifted until the idle time is completely eliminated from the schedule, generates the optimal* TWET-*makespan trade-off curve.*

Proof. In a typical optimal TWET-makespan trade-off curve, one end of the curve will correspond to the optimal TWET schedule and the other end to the optimal makespan schedule, as shown in Fig. 2. Every point lying on the TWET-Makespan trade-off curve relates to an optimal schedule for the corresponding TWET and makespan values. The makespan corresponding to a schedule in the SMSP is the completion time of the last job in the sequence (i.e. $C_{\sigma[n]}$). Therefore, any decrease in makespan from the optimal TWET schedule requires the last job in the sequence to be left shifted. Since the optimal TWET schedule is the minimum point on the TWET-$C_{\sigma[n]}$ plot, the left shifting of any job or set of jobs will increase the TWET value with the slope *SL* of the resulting TWET-makespan plot becoming a negative value. When the makespan is improved by left shifting the last job in σ, each unit of improvement in makespan must result in the optimum value (i.e. smallest possible increase in value) of TWET. The block *B*, which contains the position identifier of the job at the last position in σ as well as all its preceding set of contiguous jobs, when left shifted, results in the smallest possible increase in the value of TWET per unit decrease in makespan. This is because a job or a set of jobs preceding the jobs whose position identifiers are in *B* are already at the minimum point of their respective $TWET(\sigma_k)$ cost function as described in Theorem 1, and if left shifted along with jobs with position identifiers in *B*, it will lead to a higher rate of increase in TWET per unit decrease in makespan. Therefore, the last job and all its preceding contiguously scheduled jobs whose position identifiers are in *B*, when left shifted, results in the minimum increase in the value of TWET per unit decrease in makespan. In the process of left shifting, the position identifiers of jobs in *B* may become contiguous with a preceding job not belonging to *B*, then block *B* is updated to include the position identifiers of the job or the set of contiguous preceding jobs, resulting in a breakpoint in the TWET-$C_{\sigma[n]}$ plot with slope *SL* of the cost function becoming more negative. The slope *SL* of the TWET-$C_{\sigma[n]}$ plot also changes and becomes more negative when the jobs in the process of left shifting shift from the set *TY* to *DW* and from *DW* to *EY*. Eventually, the procedure of updating block *B* and left shifting of jobs leads to the complete removal of idle time from the schedule, resulting in the point of optimal makespan in the TWET-$C_{\sigma[n]}$ plot. Since the *SL* value decreases monotonously at each breakpoint, the optimal TWET-$C_{\sigma[n]}$ trade-off plot will always be a piecewise linear convex trade-off curve if idle time exists in the optimal TWET schedule.

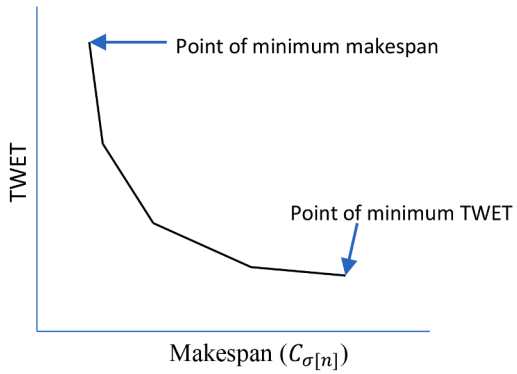


Fig. 2. A typical optimal TWET-makespan trade-off curve.

4.2. Pareto-optimal front generation procedure

The Pareto-based bi-objective metaheuristic algorithms presented in this paper use specific mechanisms to generate, perturb and improve sequences of jobs to find the Pareto-optimal solutions. Since the optimal timing algorithm (OTA) generates either a single trade-off point or a piecewise linear convex trade-off curve with infinite trade-off points on a Pareto chart corresponding to each sequence of jobs, the Pareto-optimal front generated from a given set of solutions will eventually comprise line segments and points. We adopted the Pareto-optimal front generation procedure proposed in [63] that trims and merges multiple TWET-makespan trade-off curves and trade-off points to generate the Pareto-optimal front, where each trade-off curve or point corresponds to either a single sequence of jobs or is a Pareto front comprising line segments and points generated from multiple sequences of jobs. Each line segment or point in the resulting Pareto-optimal front will be associated with a sequence of jobs, and the Pareto front generation procedure ensures that the sequences of jobs in the set related to a Pareto-optimal front are not repeated. Unlike the other scheduling problems, where each sequence of jobs results in a single trade-off point, the Pareto-optimal front generated in this paper may have a single point or one or more line segments associated with a single sequence of jobs. Fig. 3(a) shows typical TWET-makespan trade-off curves generated corresponding to multiple sequences of jobs. Some sequences of jobs may result in trade-off plots comprising a single trade-off point similar to the one corresponding to the sequence σ_4 in the figure. Fig. 3(b) shows the Pareto-optimal front generated with the non-dominated solutions from multiple sequences of jobs, as shown in Fig. 3(a). In the figure, no line segment corresponding to the sequence σ_2 lies on the Pareto-optimal front since all the line segments lying on the trade-off plot corresponding to it have been dominated by the other line segments as shown in Fig. 3 (a). Further, the job sequences σ_1 and σ_3 have resulted in multiple line segments belonging to their trade-off plots lying on the Pareto-optimal front. Readers may refer to the procedure proposed in [63] to gain a detailed understanding of the methodology. The Pareto-optimal front generation procedure generates the Pareto front such that the line segments and points on the Pareto front are arranged in the increasing order of their makespan values and also place their respective sequences of jobs in that order. The final Pareto-optimal front generated between the makespan and TWET using the metaheuristic algorithms will provide the end users with the cost proportions of compromising either of the objectives, thereby allowing them to make the best decision regarding the minimization of both objectives.

4.3. Initial solution generation

The initial solutions for the metaheuristic algorithms presented in this paper for the simultaneous minimization of the TWET and makespan objectives were generated using a heuristic methodology based on the apparent tardiness cost with setups (ACTS) rule [66–68]. The sequence generation begins with an empty set σ to which *n* unscheduled jobs are successively appended using a probabilistic rule based on heuristic desirability. The dispatching rule of ACTS is used as the heuristic desirability. The heuristic desirability η_{kb} of assigning a job *b* to position *k* in σ is defined as [69]

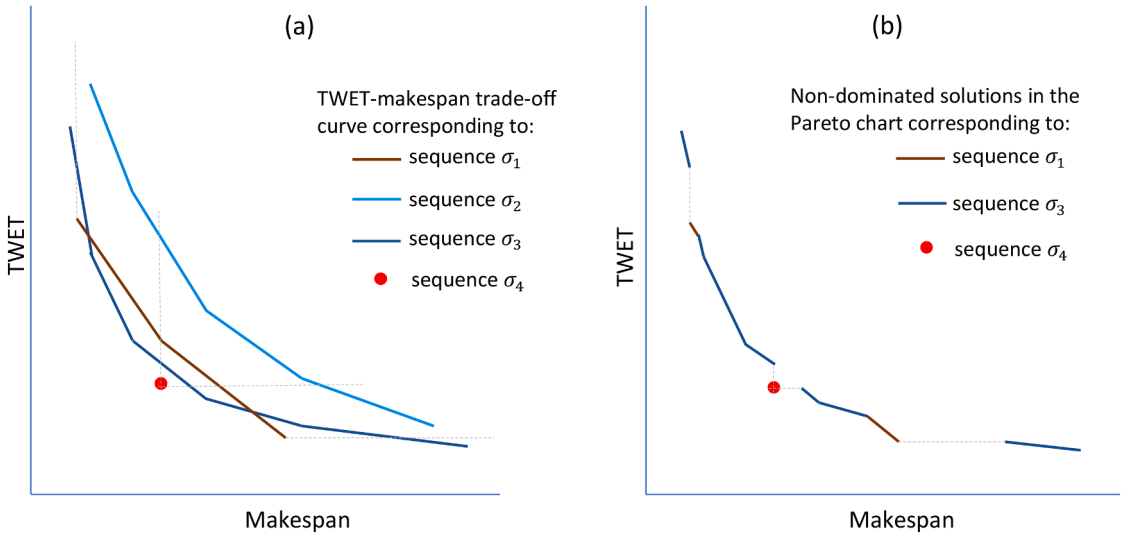


Fig. 3. (a) Typical TWET-makespan trade-off curves generated with multiple sequences of jobs, (b) Pareto-optimal front generated with the non-dominated solutions from multiple sequences of jobs.

$$\eta_{kb} = \exp\left[-\frac{de_b * k_1}{d_{avg}}\right] * exp\left[-\frac{(P_b + s_{ab}) * k_2}{s_{avg}}\right] \quad (15)$$

where de_b is the earliest due date of job b, d_{avg} is the average of the earliest due dates of all the jobs unassigned to σ, k_1 and k_2 are the scaling parameters related to the due dates and the sum of processing times and setup times $(s_{ab} + P_b)$ between the jobs, respectively. s_{ab} is the setup time between job b and the job at position $k-1$ in σ denoted by a, and s_{avg} is the average of the processing times inclusive of the setup times between the jobs unassigned to σ. The job j to be assigned at each position in the sequence is determined probabilistically based on the value of the random variable S selected according to a probability μ_{kb} defined corresponding to each unassigned job $b \in U$, as shown in Eq. (16).

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

where U denotes the set of jobs not assigned to σ. The cumulative value of μ_{kb} $\forall b \in U$ corresponding to each position k is calculated, and the job b that corresponds to the range of the random variable S will be assigned to k. This ensures fair allocation of jobs even when more than one job in U has the same value of μ_{kb}, as a job with a larger range has a higher probability of being selected. This process is performed iteratively until all the n jobs are assigned to σ. The sequences generated are further used as the initial population in the metaheuristic optimization algorithms presented in this paper.

4.4. *Multi-objective neighbourhood search-based metaheuristic algorithms*

Neighbourhood search-based metaheuristic algorithms applied to scheduling problems employ various neighbourhood structures that explore the feasible solution space by generating and evaluating the sequences in the neighbourhood of a given sequence of jobs to be improved. These algorithms can also be parallelized, allowing the use of parallel computing systems, thereby enabling them to handle complex problems within a reasonable computation time. Some of the basic neighbourhood search-based metaheuristic algorithms existing in the literature include variable neighbourhood search (VNS), variable neighbourhood descent (VND), iterated local search (ILS), Tabu search (TS), Simulated Annealing (SA), etc. These algorithms have been extensively used in both single objective and multi-objective machine scheduling problems [70–73]. Several variants of these algorithms have also been presented in the literature for the Pareto-based multi-objective optimization problems in machine scheduling, such as VNS with intensification, Pareto iterated local search, Pareto archived simulated annealing, etc. [74,75].

These algorithms, in general, begin with a single or a set of initial solutions that are generated either randomly or using heuristics. One or more neighbourhood generation mechanisms are then employed to generate solutions in the neighbourhoods of the initial solution set. Each of the neighbourhood solutions generated is then evaluated for the objectives considered in the problem. The solutions that meet the predefined acceptance criteria are selected for further exploration. Sometimes, these algorithms also accept inferior solutions with a certain probability of escaping out of the local minima in the solution space. The process of neighbourhood generation, objective evaluation and selection of solutions for further exploration is repeated iteratively until specific predefined termination criteria are met. The commonly used termination criteria include reaching a maximum number of iterations, reaching the limit on computational resources or achieving a certain level of improvement.

Apart from the neighbourhood search-based approaches, several population-based methods known to perform well for multi-objective Pareto-based optimization problems exist in the literature viz. multi-objective particle swarm optimization (MOPSO), multi-objective genetic algorithm (MOGA), multi-objective genetic local search (MOGALS), Pareto envelope-based selection algorithms (PESA and PESA-II), non-dominated sorting genetic algorithm II (NSGA-II), non-dominated ranking genetic algorithm (NRGA), etc. [76]. However, most of these methodologies use crowding distance operators [77] or region-based selection techniques [78] to evaluate solutions for dominance while generating the Pareto front. As shown in Fig. 3(b), the line segments on the Pareto front belonging to different sequences of jobs need not necessarily have a gap between them and a solution on a Pareto front consisting of line segments may span over the entire Pareto front as represented by the line segments corresponding to the sequence σ_1. Therefore, it may not be possible to determine the crowding distance of solutions or confine a solution within a region in the objective space. Hence, the existing technique of fitness evaluation from the literature cannot be implemented in this scenario. Consequently, the population-based methodologies from the literature cannot be easily adapted to solve the scheduling problem considered in this paper. This has encouraged us to experiment with neighbourhood search-based heuristic methodologies that are easily adaptable over the population-based heuristic methods in the literature.

This paper presents six Pareto-based neighbourhood search

algorithms to solve the bi-objective SMSP, which are listed below.

(1) Multi-objective variable neighbourhood descent (MOVND)
(2) Multi-objective iterated variable neighbourhood descent (MOIVND)
(3) Hybrid multi-objective variable neighbourhood descent - path relinking (MOVND_PR)
(4) Multi-objective variable neighbourhood search (MOVNS)
(5) Multi-objective variable neighbourhood search with intensification (MOVNS_I)
(6) Multi-objective iterated local search (MOILS)

The performance of the proposed VND-based algorithms is then compared with that of the VNS and ILS algorithms adopted from the literature to validate their computational efficiency. The subsequent sections of this paper first present the various neighbourhood structures used in the above algorithms. Further, the implementation of the neighbourhood search-based metaheuristic algorithms to the considered problem is presented.

4.4.1. Neighbourhood structures used in the metaheuristic algorithms

The metaheuristic algorithms presented in this paper utilise one or more of six different neighbourhood generation schemes to generate a set of neighbourhoods corresponding to a given sequence of jobs. The various neighbourhood generation mechanisms for generating neighbourhoods of a given sequence of jobs are described as follows.

(1) Swap neighbourhood (N_1): Two jobs within a fixed range of job positions in the sequence denoted by *swap_limit*, are randomly selected, and their positions in the sequence are interchanged.
(2) Insertion neighbourhood (N_2): A job is removed from the sequence and inserted back at every position in a fixed range of job positions, denoted by *shift_limit*. The *shift_limit* equally spreads over the succeeding and preceding job positions of the initial position of the removed job.
(3) 2-job insertion neighbourhood (N_3): This neighbourhood generation scheme is similar to N_2, with the exception that the jobs at two consecutive positions selected from the sequence are removed and inserted back at all positions within the range of job positions in the sequence denoted by *dshift_limit*.
(4) 3-job insertion neighbourhood (N_4): This neighbourhood generation scheme is similar to N_2, with the exception that the jobs at three consecutive positions selected from the sequence are removed and inserted back at all positions within the range of job positions in the sequence denoted by *tshift_limit*.
(5) Enumeration neighbourhood (N_5): In this neighbourhood generation mechanism, all possible combinations of job positions within a fixed range of positions in the sequence denoted by *enum_limit,* are generated, each of which contributes to one neighbourhood. The number of neighbourhoods thus generated within a fixed *enum_limit* will be $m!$, where m represents the number of jobs within the *enum_limit*. Once all possible neighbourhoods within a specific job window defined by the *enum_limit* are generated, the job window slides forward by one job position along the job sequence, and the process repeats.
(6) Compound insertion move neighbourhood (N_6): This neighbourhood generation structure uses a parameter known as search depth, denoted by d, which can either be fixed or varied during the search process to maintain a balance between exploring and exploiting the feasible solution space. This paper proposes an improved variant of the compound insertion move neighbourhood structure presented in [73]. In their paper, Xu et al. [73] randomly selected a job position r in the sequence from which d number of consecutive jobs are removed from the sequence. Let σ_R represent the set of removed jobs and σ_P represent the partial sequence comprising the jobs remaining in the sequence. The neighbourhoods are generated by iteratively inserting each removed job in σ_P at all job positions in the partial sequence σ_R. In this paper, we have proposed an improved neighbourhood structure in which d number of jobs to be removed from the sequence are selected randomly, rather than removing a set of consecutive jobs from a randomly selected position in the sequence.

The multi-objective metaheuristic algorithms presented in this paper employ one or more of the above neighbourhood generation mechanisms.

4.4.2. The proposed MOVND algorithm

The multi-objective variable neighbourhood descent (MOVND) algorithm proposed in this paper is inspired by the sequential variable neighbourhood descent algorithm presented in [79]. We have suitably adapted the methodology presented originally for a single objective optimization problem to extend it to the multi-objective optimization problem considered in this paper. According to the methodology presented in [79], whenever an improved solution is obtained with a neighbourhood generation mechanism, the neighbourhood generation mechanism is reset to the first one. Whenever there is no improvement in solutions with any of the neighbourhood generation mechanisms, the neighbourhood generation mechanism is sequentially switched to the next in a pre-defined order.

Algorithm 3 shows the procedure for the proposed MOVND algorithm that begins by generating a Pareto-optimal front of the solutions in the initial population P, denoted by A, as shown in steps 1–2. Step 2 involves generating a piecewise linear convex trade-off curve between the two objectives corresponding to each sequence of jobs in P using the proposed optimal timing algorithm (OTA) described in Section 4.1 and further generating the Pareto-optimal front of the trade-off curves generated corresponding to all the sequences in P using the procedure described in Section 4.2. The sequences corresponding to the line segments belonging to the Pareto-optimal front A are then subjected to one or more neighbourhood generation mechanisms described in Section 4.4.1.

The proposed MOVND algorithm utilizes k_{max} number of neighbourhood generation mechanisms represented by the ordered set $N^{VND} \subseteq \{N_1, N_2, N_3, N_4, N_5\}$ to generate the neighbourhood solutions in the procedure, such that $k_{max}=|N^{VND}|$. The binary variable $visited\_flag(\sigma,\ k) \in \{0,1\}$ denotes whether each sequence $\sigma \in A$ has been subjected to neighbourhood generation using the neighbourhood structure k, where $k=1,2,\ldots,k_{max}$. $visited\_flag(\sigma,\ k)\ \forall\sigma \in A$ are marked as unexplored initially $\forall k$, as shown in step 3. Each solution $\sigma \in A$ is then improved iteratively by generating all possible neighbourhoods using the neighbourhood structures one after the other, as shown in steps 4–21. M denotes the set of neighbourhood solutions generated for the sequences of jobs $\sigma \in A$ using the chosen neighbourhood structure in the set N^{VND}, as shown in step 7. The neighbourhoods are generated only for those sequences of jobs in A that were previously unexplored by the chosen neighbourhood structure k. $visited\_flag(\sigma,\ k)\ \forall\sigma \in A$ generated using the neighbourhood structure $N^{VND}(k)$ is marked as visited, as shown in step 8. The Pareto-optimal front of the solutions in M is then generated, denoted by A', and the $visited\_flag(\sigma,\ k)\ \forall\sigma \in A'$ is set as unvisited $\forall k$, as shown in steps 9 and 10, respectively. Similar to Step 2, Step 9 involves the generation of trade-off curves between the objectives corresponding to each sequence of jobs in M using the OTA and the subsequent generation of the Pareto-optimal front of the trade-off curves generated corresponding to all the sequences in M. The non-dominated solutions from A' update the Pareto-optimal front A, as shown in step 15. If the Pareto-optimal front A has improved in this process, then the neighbourhood structure is reset to $N^{VND}(1)$ as shown in step 17, and steps 6–15 are repeated. On the other hand, if the Pareto-optimal front A has not improved by the generation of neighbourhoods of sequences $\sigma \in$

Algorithm 3
The MOVND algorithm.

```
1  P ← InitialPopulationGeneration()
2  A ← ParetoOptimalFrontGeneration(P)
3  visited_flag(σ, k) ← false ∀σ ∈ A, ∀k
4  k ← 1, , k_max ← |N^VND|
5  while (CPU_Time < CPU_TimeLimit) do
6      if (k ≤ k_max) then
7          M ← GenerateNeighbourhoods(σ, N^VND(k)) ∀σ ∈ A: visited_flag(σ, k)=false
8          visited_flag(σ, k) ← True ∀σ ∈ A
9          A' ← ParetoOptimalFrontGeneration(M)
10         visited_flag(σ, k) ← false ∀σ ∈ A', ∀k
11     else
12         σ ← SelectJobSequence(A)
13         A' ← Intensification(σ, d)
14     end
15     A ← ParetoOptimalFrontUpdate(A ∪ A')
16     if (is_improved(A) = true) then
17         k ← 1
18     else if (k ≤ k_max) then
19         k ← k + 1
20     end
21 end
```

A generated using a particular $N^{VND}(k)$, then, the following neighbourhood structure is chosen to further explore the neighbourhoods of the sequences in A, as shown in step 19. If all the neighbourhood structures in the ordered set N^{VND} have been explored, and no further improved solution is found in A, then set A is subjected to intensification (N_6), as shown in steps 12 and 13. Intensification is an improvement mechanism proposed in [13], which involves randomly selecting a sequence $\sigma \in A$ and exploring the solution space for an improved solution in its neighbourhood. Algorithm 4 shows the procedure of intensification.

Intensification uses the neighbourhood structure N_6 described in Section 4.4.1, which begins with removing d number of jobs from $\sigma \in A$. σ_R denotes the set of removed jobs and σ_P denotes the partial sequence

Algorithm 4
Intensification.

```
1  Function Intensification(σ ,d)
2      σ_P, σ_R ← N_6(σ, d)
3      B' ← ParetoOptimalFrontGeneration(σ_p)
4      for (k = 1 to d) do
5          B'' ← ∅
6          for ∀σ_q ∈ B' do
7              SP_σq ← ∅
8              for (p = 1 to j − d + k) do
9                  SP_σq ← partial sequences obtained by inserting job σ_R(k) at each position p in σ_q
10             end
11             B'' ← ParetoOptimalFrontUpdate(B'' ∪ SP_σq)
12         end
13         B' ← B''
14     end
15     return B'
16 end
```

obtained with the remaining $n-d$ jobs, as shown in step 2 of Algorithm 4. The TWET-makespan trade-off curve corresponding to the partial sequence σ_P is then generated and subjected to the Pareto front generation procedure to generate the Pareto-optimal front, which is denoted by B' as shown in step 3. In each iteration of steps 4–14, the k^{th} $\{k=1,2...d\}$ job in σ_R is inserted at each position p $\{p=1,2...,j-d+k\}$ of each partial sequence $\sigma_q \in B'$. Let SP_{σ_q} denote the set of partial sequences obtained with the k^{th} job added at each position p of a particular sequence $\sigma_q \in B'$, as shown in steps 8–10. A Pareto-optimal front corresponding to the TWET-makespan trade-off curves of the partial sequences belonging to SP_{σ_q} is then generated, which is denoted by B'', as shown in step 11. After the k^{th} job is inserted into every position p of every sequence $\sigma_q \in B'$, the Pareto-optimal front B'' replaces the Pareto-optimal front B', as shown in step 13. Steps 5–13 repeat until every job belonging to set σ_R has been inserted into every position p of every sequence $\sigma_q \in B'$. On intensification, if an improved solution is found, the neighbourhood structure is reset to $N^{VND}(1)$ and steps 6–15 of Algorithm 3 are repeated. If an improved solution is not found, then another sequence $\sigma \in A$ is chosen at random and subjected to intensification. To allow for a fair performance comparison between the MOVND and the other algorithms, a predefined maximum computation time (*CPU_TimeLimit*) allowed to run a specific problem instance is set as the termination criterion for all the algorithms.

4.4.3. The proposed MOIVND algorithm

The proposed multi-objective iterated variable neighbourhood descent (MOIVND) algorithm is similar to the proposed MOVND, except that in the MOIVND we have used a perturbation phase instead of the intensification used in the MOVND. Algorithms 5 and 6 show the pseudocode of the proposed MOIVND algorithm.

The MOIVND algorithm utilizes k_{max} number of neighbourhood generation mechanisms represented by the ordered set $N^{IVND} \subseteq \{N_1, N_2, N_3, N_4, N_5\}$ to generate the neighbourhood solutions in the procedure, where $k_{max}=|N^{IVND}|$. Similar to the MOVND algorithm, the MOIVND algorithm begins by generating a Pareto-optimal front of the solutions in the initial population P, denoted by A, as shown in steps 1–2. *visited_flag* (σ, k) $\forall \sigma \in A$ are marked as unexplored initially $\forall k: k=1,2,\ldots,k_{max}$, as shown in step 3. A is assigned to A', and each solution $\sigma \in A'$ is then improved iteratively by generating all possible neighbourhoods using the neighbourhood structures in the ordered set N^{IVND} one after the other, as shown in steps 4–22. Every time an improved solution is found in A', the neighbourhood structure is reset to $N^{IVND}(1)$, and if an improved solution is not found, the next neighbourhood structure in the set N^{IVND} is chosen to improve the solutions, as shown in steps 12–15. *visited_flag*(σ, k) is marked as explored if a sequence of jobs $\sigma \in A'$ has already been improved with a neighbourhood structure k, and only the sequences of jobs unexplored with certain neighbourhood structures are considered for the generation of neighbourhoods. After a set of neighbourhoods represented by M are generated and the corresponding Pareto-optimal front A' is updated, the Pareto-optimal front A is updated with A', as shown in steps 10–11. If no improved solution is found in A' after implementing all four neighbourhood structures, then the solutions in set A are perturbed by invoking the *PERTURB* function, as shown in step 17 of Algorithm 6, which generates a new Pareto front A'. The Pareto front A is updated with A' as shown in step 19, and the solutions in A' are further subjected to improvement using the four neighbourhood structures as shown in steps 6–10. The above procedure is performed until the termination criterion of a pre-specified computation time limit (*CPU_TimeLimit*) is reached.

The perturbation mechanism used in the MOIVND algorithm uses a neighbourhood generation scheme to escape the local minima and backtrack the solutions rather than relying on random perturbation moves. The procedure for the perturbation of solutions in A is shown in Algorithm 6. Initially, the solutions in A are assigned to set B, as shown in step 2. Each sequence $\sigma \in B$ is then perturbed, which involves randomly selecting a position q in the sequence and the job in that position is inserted at all other possible positions in the sequence using the neighbourhood structure N_2 to generate $n-1$ neighbourhood solutions, which is denoted by M. The Pareto-optimal front B' is then generated with the solutions in M, and the Pareto front B'' is subsequently updated with B'. After all the sequences in B are subjected to the above procedure, the solutions in B'' are assigned to B as shown in step 11. The perturbation and Pareto front update procedure, as shown in steps 6–9, is then repeated with the solutions in B, and this process continues for *perturb_iter* number of iterations. The perturbation phase allows the solutions in A to escape local minima, and the procedure of generation of the neighbourhoods followed by the generation of the Pareto-optimal front, as shown in steps 7–9, ensures that the perturbed solutions do not deviate too much from their original solutions in terms of the objective values.

4.4.4. The proposed hybrid MOVND_PR algorithm

The proposed multi-objective hybrid variable neighbourhood descent – path relinking (MOVND_PR) algorithm is similar to the proposed MOIVND algorithm, except that the MOVND_PR algorithm uses a path relinking-based strategy to explore the neighbourhoods of the solutions in the Pareto archive instead of the backtrack-based perturbation used in the MOIVND. Therefore, the procedure shown in Algorithm 5 also applies to the MOVND_PR algorithm, except that step 17 of Algorithm 5 invokes $A' \leftarrow PATH\_RELINKING(A)$. Path relinking is typically used as an intensification strategy to explore paths connecting elite solutions obtained by other metaheuristic methods, such as scatter search, GRASP, tabu search, etc. [80–83]. In this paper, the path relinking strategy is hybridized with the VND algorithm to generate new solutions in the neighbourhoods of the elite solutions on the Pareto archive, which is further improved using the local search procedure within the VND framework.

Algorithm 7 shows the path relinking phase of the MOVND_PR algorithm, which obtains the Pareto front A from Algorithm 5 and returns the Pareto front B generated using the path relinking procedure. As shown in steps 3–13 of Algorithm 7, the procedure selects each solution (i.e. the sequence of jobs) in the Pareto front A as an initial solution (s_o) and iteratively identifies the job insertions required to sequentially move it towards a randomly chosen guiding solution (s_g) belonging to A. At first, the difference in the position of each job in the two job sequences s_o and s_g is determined. As shown in steps 6–9, the job i with the highest positional difference (PD) in s_o is identified and shifted to a position number in its sequence, which is the same as the position number (PS) of this job in s_g. A job's positional difference is the absolute value of the difference of its position number in s_o with its position number in s_g. The new sequence s_{new} thus generated along with its corresponding TWET-makespan trade-off curve is assigned to (or used to update) the Pareto front B, as shown in step 10 of Algorithm 7. The new sequence s_{new} is then assigned to s_o (i.e. $s_o \leftarrow s_{new}$), as shown in step 11 of Algorithm 7. Subsequently, the job with the highest positional difference in s_o with respect to s_g is again identified and shifted to a position in s_o, which is the same as the position of this job in s_g. The new sequence s_{new} generated, along with its TWET-makespan trade-off curve, updates B as described above. The above procedure is repeated until the maximum positional difference of a job in s_o with respect to s_g is equal to or greater than D_{max}, where $D_{max}>0$. A value of D_{max} closer to zero indicates that towards the end of the above procedure of generating the sequence of neighbourhoods while moving from s_o to s_g, the solutions generated will be closer to s_g, leading to faster convergence in the subsequent local search phase shown in Algorithm 5. On the other hand, a higher value of D_{max} may lead to a higher exploration.

4.4.5. The MOVNS algorithm

We have adopted the multi-objective variable neighbourhood search (MOVNS) algorithm presented in [74], as shown in Algorithm 8.

Algorithm 5
The MOIVND algorithm.

```
1  P ← InitialPopulationGeneration()
2  A ← ParetoOptimalFrontGeneration(P)
3  visited_flag(σ, k) ← false ∀σ ∈ A, ∀k
4  k ←1, A' ← A, k_max ← |N^IVND|
5  while (CPU_Time < CPU_TimeLimit) do
6      M ← GenerateNeighbourhoods(σ, N^IVND(k)), ∀σ ∈ A': visited_flag(σ, k) = false
7      visited_flag(σ, k) ← True ∀σ ∈ A'
8      A'' ← ParetoOptimalFrontGeneration(M)
9      visited_flag(σ, k) ← false ∀σ ∈ A'', ∀k
10     A' ← ParetoOptimalFrontUpdate(A' ∪ A'')
11     A ← ParetoOptimalFrontUpdate(A ∪ A')
12     if (is_improved(A') = true) then
13         k ←1
14     else if (k ≤ k_max − 1) then
15         k ← k +1
16     else if (k = k_max) then
17         A' ← PERTURB(A)
18         visited_flag(σ, k) ← false ∀σ ∈ A', ∀k
19         A ← ParetoOptimalFrontUpdate(A ∪ A')
20         k ←1
21     end
22 end
```

Algorithm 6
Perturbation phase in the MOIVND algorithm.

```
1  Function PERTURB(A)
2      B ← A
3      for (k = 1 to perturb_iter) do
4          B'' ← ∅
5          for (∀σ ∈ B) do
6              q ← SelectPositionInSequence(σ)
7              M ← GenerateNeighbourhoods(σ, q, N_2)
8              B' ← ParetoOptimalFrontGeneration(M)
9              B'' ← ParetoOptimalFrontUpdate(B'' ∪ B')
10         end
11         B ← B''
12     end
13     return B
14 end
```

The methodology begins with an initial population of sequences denoted by P, as shown in step 1 of Algorithm 8. Each individual in the population is a sequence of n number of jobs to be scheduled, denoted by σ, as described in Section 4.1, which is generated using the procedure described in Section 4.3. A Pareto-optimal front of the TWET-makespan trade-off curves corresponding to the sequences of jobs in P is first generated, which is denoted by A, as shown in step 2. A variable neighbourhood search is then performed on each of the non-dominated solutions in A in search of better solutions in the objective space, as shown in steps 3–18. In the algorithm, the binary variable $visited\_flag(\sigma) \in \{0, 1\}$ denotes whether each sequence $\sigma \in A$ has been subjected to the neighbourhood search or not. $visited\_flag(\sigma), \forall \sigma \in A$ are marked as unvisited initially, as shown in step 3.

Each iteration of the MOVNS algorithm begins with the random selection of an unvisited solution $\sigma \in A$ as shown in step 5. Every time a solution $\sigma \in A$ is selected, $visited\_flag(\sigma)$ is marked as visited. A set of neighbourhood generation structures represented by $N^{VNS} \subseteq \{N_1, N_2, N_3, N_4, N_5\}$, described in Section 4.4.1, have been employed in this

Algorithm 7
Path relinking phase in the MOVND_PR algorithm.

```
1  Function PATH_RELINKING(A)
2      B ← ∅
3      for (∀σ ∈ A) do
4          s_o ← σ, s_g ← RandomSelectionOfSequence(A)
5          do
6              i ← IdentifyJobWithMaxPositionalDifference(s_o, s_g)
7              PS ← IdentifyPositionNumberOfJob(i, s_g)
8              PD ← ValueOfPositionalDifference(i, s_o, s_g)
9              s_new ← GenerateNeighbourhoodByJobInsertion(s_o, i, PS)
10             B ← ParetoOptimalFrontUpdate(B ∪ s_new)
11             s_o ← s_new
12         while(PD ≥ D_max)
13     end
14     return B
15 end
```

Algorithm 8
The MOVNS algorithm.

```
1  P ← InitialPopulationGeneration()
2  A ← ParetoOptimalFrontGeneration(P)
3  visited_flag(σ) ← false ∀σ ∈ A
4  while (CPU_Time < CPU_TimeLimit) do
5      σ ← SelectJobSequence(A) ∀σ ∈ A: visited_flag(σ) = false
6      visited_flag(σ) = true
7      N_k ← RandomSelection {N^VNS}
8      M ← GenerateNeighbourhoods(σ, N_k)
9      σ ← SelectJobSequence(M)
10     A′ ← ∅
11     M′ ← GenerateNeighbourhoods(σ, N_k)
12     A′ ← ParetoOptimalFrontGeneration(M′)
13     visited_flag(σ) ← false ∀σ ∈ A′
14     A ← ParetoOptimalFrontUpdate(A ∪ A′)
15     if(visited_flag(σ) = true ∀σ ∈ A) then
16         visited_flag(σ) ← false ∀σ
17     end
18 end
```

algorithm, from which one structure is randomly selected in every iteration of the search, denoted by N_k, as shown in step 7. All possible neighbourhoods of $\sigma \in A$ are then generated using N_k, and one of the neighbourhood solutions generated is randomly selected for further exploration, as shown in steps 8–9. Let M represent the set of all neighbourhoods of $\sigma \in A$ generated using N_k. A sequence $\sigma \in M$ is randomly selected, and all possible neighbourhoods of $\sigma \in M$ are generated using N_k. Let M' denote the set of all neighbourhood solutions hence generated. A Pareto-optimal front of all the sequences in M' is further generated, which is denoted by A', as shown in steps 10–12. *visited_flag*(σ), $\forall \sigma \in A'$ is initialized as unvisited, as shown in step 13. The Pareto-optimal front A is then updated with the non-dominated solutions in the Pareto-optimal front A', as shown in step 14. Once every $\sigma \in A$ is visited, *visited_flag*(σ) is marked as unvisited, as shown in steps 15–17. Steps 5–17 are repeated until the *CPU_TimeLimit* is reached.

4.4.6. The MOVNS_I algorithm

We have adopted the multi-objective variable neighbourhood search algorithm with intensification (MOVNS_I) presented in [74], as shown in Algorithm 9. This method is an improvement over the basic MOVNS

Algorithm 9
The MOVNS_I algorithm.

1 $P \leftarrow InitialPopulationGeneration()$
2 $A \leftarrow ParetoOptimalFrontGeneration(P)$
3 $visited\_flag(\sigma) \leftarrow false \ \ \forall \sigma \in A$
4 **while** $(CPU\_Time < CPU\_TimeLimit)$ **do**
5 $\quad \sigma \leftarrow SelectJobSequence(A) \ \forall \sigma \in A: visited\_flag(\sigma) = false$
6 $\quad visited\_flag(\sigma) \leftarrow true$
7 $\quad N_k \leftarrow RandomSelection \ \{N^{IVNS}\}$
8 $\quad M \leftarrow GenerateNeighbourhoods(\sigma, N_k)$
9 $\quad \sigma \leftarrow SelectJobSequence(M)$
10 $\quad M' \leftarrow GenerateNeighbourhoods(\sigma, N_k)$
11 $\quad A' \leftarrow \emptyset$
12 $\quad A' \leftarrow ParetoOptimalFrontGeneration(M')$
13 $\quad \sigma \leftarrow SelectJobSequence(A')$
14 $\quad A'' \leftarrow Intensification \ (\sigma, d)$
15 $\quad visited\_flag(\sigma) \leftarrow false \ \ \forall \sigma \in (A' \cup A'')$
16 $\quad A \leftarrow ParetoOptimalFrontUpdate(A \cup A' \cup A'')$
17 $\quad$ **if** $(visited\_flag(\sigma) = true \ \forall \sigma \in A)$ **then**
18 $\quad\quad visited\_flag(\sigma) \leftarrow false \quad \forall \sigma$
19 $\quad$ **end**
20 **end**

algorithm described in Section 4.4.5. Steps 1–12 and 17–19 of Algorithm 9 are similar to steps 1–12 and 15–17 of Algorithm 8. $N^{IVNS} \subseteq \{N_1, N_2, N_3, N_4, N_5\}$ represents the set of neighbourhood structures employed for generating neighbourhoods in the MOVNS_I algorithm. In step 13 of Algorithm 9, a job sequence $\sigma \in A'$ is randomly selected and subjected to the intensification procedure as shown in Algorithm 4 of Section 4.4.2.

At the end of the intensification procedure, the sequences of jobs in B' replaces A'' as shown in step 14 of Algorithm 9. Further, the Pareto optimal front A is updated with the non-dominated solutions from the Pareto optimal fronts A' and A'' as shown in step 16 of Algorithm 9.

4.4.7. The MOILS algorithm

We have adopted the multi-objective iterated local search (MOILS) algorithm proposed in [73]. They implemented a Pareto-based local search method to improve the solutions, which is essentially the same as the intensification procedure proposed in [13] as described in Algorithm 4. However, the search depth d used in the intensification is varied throughout the search process in the MOILS algorithm, unlike the MOVNS algorithm described in the previous section, which uses a constant search depth. Further, the neighbourhood structure N_6, which is an improvement over the neighbourhood generation mechanism presented in [73], as described in Section 4.4.1, is employed to generate the neighbourhoods during intensification.

Algorithm 10 shows the MOILS procedure. It begins with the generation of the Pareto-optimal front of the initial set of sequences of jobs in P, which is denoted by A, as shown in step 2. A sequence $\sigma \in A$ is then randomly selected for exploration and subjected to the intensification procedure with an initial search depth d_0, as shown in steps 3–4. Let L denote the set of improved solutions obtained as a result of intensification using d_0, the non-dominated solutions of which update the Pareto-optimal front A as shown in step 5. The binary variable $update\_flag \in \{0, 1\}$ in the algorithm denotes whether the Pareto-optimal front A has improved or not after each iteration of the local search. Each iteration of the MOILS algorithm consists of a perturbation and an intensification procedure, as shown in steps 7–25. A sequence $\sigma \in A$ is randomly selected and perturbed using one random iteration of the insertion neighbourhood generation N_2 described in Section 4.4.1, as shown in steps 8 and 11, respectively, in Algorithm 10. *update_flag* is initialized to *true* at the beginning of the iterated local search, as shown in step 9. The search depth d is varied within the range 1–5 in consecutive iterations of the local search, as shown in steps 12–16. The perturbed sequence σ' obtained at step 11 is subjected to an intensification procedure with the updated search depth d, as shown in step 17. The non-dominated solutions from the set L of improved solutions, hence obtained, update the Pareto-optimal front A as shown in step 18. Whenever an improved sequence σ' is added to the Pareto-optimal front A, σ' replaces the current best solution σ as shown in step 20. The steps 11–23 are repeated until the Pareto-optimal front A has no improvement. Otherwise, the *update_flag* is set as false, as shown in steps 21–23. The procedure shown in steps 8–24 is repeated until the predefined limit on the computation time (*CPU_TimeLimit*) is reached.

5. Computational study

This section presents the computational study of the neighbourhood search-based metaheuristic algorithms presented in this paper, using a set of 24 problem instances with the number of jobs varying from 20 to 100. The algorithms were programmed in C language and run using the Intel *C++* Compiler version 2022.2.1 on a 2.6 GHz Intel Xeon Gold 6132 dual processor workstation with 28 cores, 128 GB RAM, and Linux OS. The correctness of the C program of the proposed optimal timing algorithm (OTA) embedded within the metaheuristics was verified by modelling and solving the mathematical formulation presented in Section 3, using IBM ILOG CPLEX ver 12.7.1 callable libraries [84]. The decision variable relating to a fixed sequence of jobs (x_{ij}) was input into the mixed integer linear programming (MILP) model and the solution obtained using CPLEX (i.e., C_i, E_i, T_i) and their objective values were compared with those obtained using the proposed OTA to verify the results. The makespan values obtained at every breakpoint on the TWET-makespan trade-off curve were input to the MILP model that

Algorithm 10
The MOILS algorithm.

```
P ← InitialPopulationGeneration()
A ← ParetoOptimalFrontGeneration(P)
σ ← SelectJobSequence(A)
L ← Intensification(σ, d_0)
A ← ParetoOptimalFrontUpdate(A ∪ L)
d ← d_0
while (CPU_Time < CPU_TimeLimit) do
    σ ← SelectJobSequence (A)
    update_flag ← true
    while (update_flag = true & CPU_Time < CPU_TimeLimit) do
        σ′ ← RandomInsertion(σ)
        if (d > 1) then
            d ← d − 1
        else
            d ← d_0
        end
        L ← Intensification(σ′, d)
        A ← ParetoOptimalFrontUpdate(A ∪ L)
        if (is_improved (A) = true) then
            σ ← σ′
        else
            update_flag ← false
        end
    end
end
```

calculates the corresponding optimal TWET to verify the correctness. The neighbourhood generation procedure in the metaheuristic algorithms was parallelized for multi-core processing using OpenMP [85]. Since the arithmetic operations involving floating point variables in the Pareto-optimal front generation method are prone to errors [63,86], the related flag (*-fp-model*) was set to "strict" while compiling the code to ensure the highest floating point precision. The flag relating to the optimization level was set to 0 (i.e. -O0), which disables all forms of optimization during the execution of the code.

This section first presents the procedure adopted from the literature for the generation of test instances and the parameter settings used in implementing the metaheuristic algorithms. Further, the performance evaluation metrics used to compare the performance of the proposed metaheuristic algorithms are presented. Subsequently, the performance comparison of the metaheuristic algorithms using different performance evaluation metrics is presented. Finally, a few insights from the parameter settings using multiple trials of the best-performing algorithm have been presented.

5.1. Generation of test instances

To the best of our knowledge, there exists no known benchmark data in the literature for the multi-objective single machine scheduling problem considered in this paper. Therefore, the problem instances were generated using the procedure presented in [13]. Let n denote the number of jobs selected from the set {20, 30, 40, 50, 75, 100}. The processing time and the tardiness penalty for each job were uniformly generated in the intervals [1, 100] and [20, 100], respectively and the earliness penalty for each job was generated as k times the tardiness penalty of the job, where k denotes a random number generated in the interval [0,1]. The centre of the due window $[de_i, dt_i]$ corresponding to a job i was uniformly generated in the interval $[(1-T-RDD/2)\,TP, (1-T+RDD/2)TP]$, where TP denotes the total processing time for all the jobs in the sequence, T denotes the tardiness factor, and RDD denotes the relative range of the due windows. The values of T and RDD are selected respectively from the sets {0.1, 0.3} and {0.8, 1.2} and the sizes of the due windows were distributed uniformly in the interval $[1, TP/n]$. The setup time $S_{ii'}$ between each pair of jobs (i, i'): $i \neq i'$, were uniformly generated in the range [0,50]. For each n, 4 test instances were generated with each pair of (T, RDD), thereby generating $4\times6=24$ problem instances. The test instances follow the naming convention *Pn_Tp_Rq*, where p and q denote the T and RDD values, respectively.

5.2. Parameter settings

The parameters in the metaheuristic algorithms were fine-tuned through trials. The algorithms were run ten times for a selected set of problem instances to find the best settings for each parameter. Subsequently, to compare the performance of the algorithms, the parameter values that delivered the best solution quality and computational efficiency on the selected set of instances were further used for the performance study, in which the algorithms were run ten times for each problem instance. The best settings found for various parameters are listed below.

- Parameters in the initial solution generation: The scaling parameters k_1 and k_2 used in the heuristic methodology were randomly generated in the range [1,10] to generate the initial solutions.
- Search depth in the MOVNS_I and MOVND algorithms: The search depth d was set to 4 based on the settings used in the literature [74].
- Search depth in the MOILS algorithm: The initial search depth $d = d_0$ was set to 5, and further, the search depth d varied between 1 and 5 cyclically with each iteration of the search [73].
- Perturbation parameter (*perturb_iter*) in the MOIVND algorithm: The perturbation factor was set as *perturb_iter*=10.
- D_{max} in the MOVND_PR algorithm: $D_{max} = 3$
- Termination criterion: For all the algorithms, the termination criterion was set to *CPU_TimeLimit* = 50 x n seconds for $n \in \{20,30\}$ and *CPU_TimeLimit* = 100 x n seconds for $n \in \{40,50,75,100\}$, where n denotes the number of jobs in the problem instance.
- Parameters in the neighbourhood generation: The best combination of neighbourhood structures for N^{VND}, N^{IVND}, N^{VNS}, N^{IVNS} were found to be $N^{VND} = \{N_1, N_2, N_3, N_4, N_5\}$, $N^{IVND} = N^{VNS} = N^{IVNS} = \{N_1, N_2, N_3, N_4\}$. For the neighbourhood generation in the MOVND, MOIVND, MOVND_PR, MOVNS, MOVNS_I and MOILS algorithms, the parameters set were: *swap_limit* = 30, *shift_limit* = 15, *dshift_limit* = 15, *tshift_limit* = 15, *enum_limit* = 6. However, these limits, except *enum_limit,* were not imposed on the neighbourhood generation of the first *seq_wl* number of sequences of jobs in the Pareto-optimal front, the reason for which is explained below.

The optimization of the TWET objective causes the jobs in the schedule to be clustered around their corresponding earliest due dates to minimize the earliness or tardiness penalties. This implies that the optimization of TWET necessitates the generation of neighbourhoods only within a range of job positions so that the job completion times are in proximity to their respective earliest due dates. Therefore, during the neighbourhood generation, *swap_limit, shift_limit, dshift_limit, tshift_limit* and *enum_limit* are imposed on the corresponding neighbourhood generation mechanisms that restrict the range of job positions in the sequences within which the jobs can be repositioned. This limit on the neighbourhood generation helps to minimize the computation time required to generate neighbourhood solutions, their respective TWET-makespan trade-off curves, and their Pareto-optimal front. However, optimising the makespan necessitates more exploration of the solution space, requiring the jobs to be repositioned to any position in the sequence. As discussed in Section 4.2, the sequences of jobs corresponding to the non-dominated solutions on the Pareto-optimal front are arranged in the increasing order of makespan values or the decreasing order of TWET values. Therefore, the metaheuristic algorithms were allowed to generate all possible neighbourhoods corresponding to the first *seq_wl* sequences of jobs on the respective Pareto-optimal fronts with no restrictions on the range of job positions in the sequence between which the jobs can be repositioned, except the *enum_limit*. The remaining sequences of jobs were subjected to the neighbourhood generation using the limits (*swap_limit, shift_limit, dshift_limit, tshift_limit* and *enum_limit*) to minimize the computational effort. The best value of *seq_wl* was found to be 30 based on fine-tuning a selected set of test instances.

5.3. Performance metrics

The literature discusses several performance metrics to compare the solution sets generated by multi-objective optimization algorithms [87,88]. Most of the metrics in the literature are proposed to compare Pareto-optimal fronts, which are comprised of only points. However, the Pareto-optimal fronts generated by the metaheuristic algorithms presented in this paper comprise line segments and points. Therefore, we have suitably adapted four performance metrics from the literature to compare the Pareto-optimal fronts generated by the metaheuristic algorithms, which are described as follows.

5.3.1. Hypervolume

Hypervolume is a performance metric that measures the area enclosed between the non-dominated solution set, obtained by an optimization algorithm whose efficiency is to be measured, and the boundaries set on the objective values corresponding to a reference point in the objective space. The calculation of hypervolume for a Pareto-optimal front comprising points is described in [89]. We have adapted their methodology to calculate the hypervolume for a Pareto-optimal front comprising line segments and points.

Fig. 4(a) shows a Pareto-optimal front between two objectives comprising line segments and points. In the figure, the solid line segments and points represent the non-dominated solutions in the Pareto-optimal front. P_{ref} denotes the reference point, the intercepts from which generate the boundaries on the axes corresponding to the bi-objectives, as shown by the dotted lines in Fig. 4(a). The hatched area enclosed by these boundaries and the Pareto-optimal front represents the hypervolume achieved by the Pareto-optimal front. To calculate the hypervolume, the area enclosed is obtained by dividing the hatched area into rectangles and triangles, as shown in Fig. 4(b), and then adding up their areas. For each problem instance, the metaheuristic algorithm achieving the highest value of hypervolume is the best in terms of convergence [90].

5.3.2. Averaged hausdorff distance

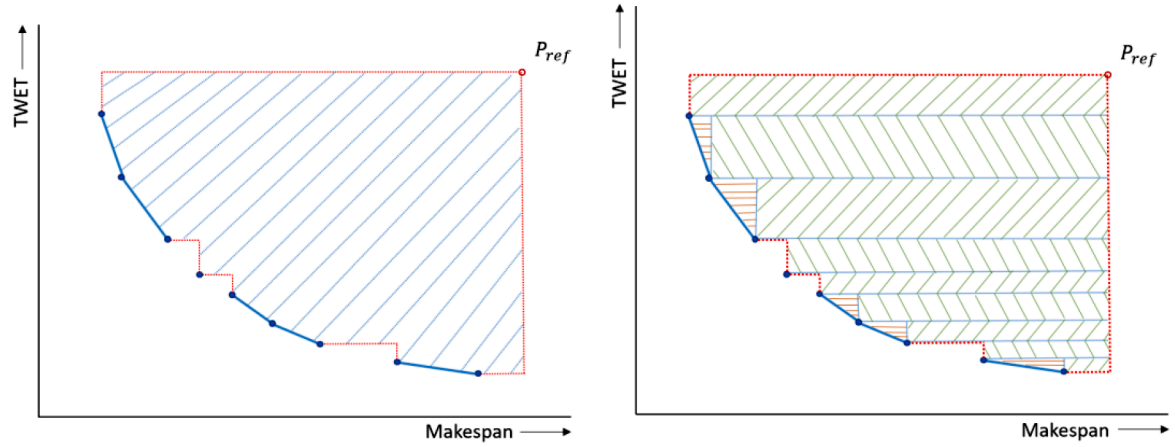
While evaluating the effectiveness of optimization algorithms, in addition to Pareto-optimality, several other aspects determine the quality of the approximation, such as convergence, spread and distribution [91]. Averaged Hausdorff distance is a widely used performance metric that measures these three aspects concurrently. It is an averaged measure of two other performance metrics, namely generational distance and inverted generational distance, originally proposed for comparing Pareto-optimal fronts comprising points [92]. In this paper, the Pareto-optimal fronts comprising line segments and points have been discretized into points to adopt this metric.

Generational distance (GD) was first proposed in [93] to determine the efficiency of multi-objective evolutionary algorithms. It measures the closeness of a Pareto-optimal front comprising points to a known true Pareto-optimal front. Let P denote the Pareto optimal front obtained by a metaheuristic algorithm whose efficiency is to be evaluated, and Q denote the known true Pareto-optimal front. Then, the generational distance is calculated as [92]

$$GD_p(P,Q) = \frac{\left(\sum_{i=1}^{|P|} dist(p_i,Q)^p\right)^{\frac{1}{p}}}{|P|} \tag{17}$$

where $|P|$ denotes the number of non-dominated solutions on P, and $dist(p_i, Q)$ denotes the Euclidean distance between each solution point p_i on P and the nearest point to p_i on Q. In this paper, the known true Pareto front Q termed the reference Pareto, is the Pareto-optimal front of the Pareto fronts generated by all the runs of all the employed metaheuristic algorithms. The GD is then calculated between the discretized Q and the discretized Pareto-optimal front generated in each run of the metaheuristic algorithms. The gaps between the line segments and points in Q are interpolated with points while discretizing them.

A minimum value of GD implies that the algorithm has a better convergence to Q [91,94]. However, this metric is largely influenced by outliers as it measures only the convergence of the solutions and not the coverage [89,91]. This means that a few outliers on P will increase GD, making the algorithm appear inferior even if it has found a substantial number of optimal solutions. To overcome the limitations of GD, inverted generational distance (IGD) was proposed in [95]. IGD is a metric similar to GD, except that P and Q are interchanged in Eq. (17) as [92]



(a) Representation of hypervolume (b) Calculation of hypervolume

Fig. 4. Calculation of hypervolume for a Pareto-optimal front comprising line segments and points.

$$IGD_p(Q,P) = \frac{\left(\sum_{i=1}^{|Q|} dist(q_i,P)^p\right)^{\frac{1}{p}}}{|Q|} \tag{18}$$

where $|Q|$ denotes the number of non-dominated solutions on Q, and $dist(q_i,P)$ denotes the Euclidean distance between each solution point q_i on Q and the nearest point to q_i on P. This metric ensures that the spread and distribution aspects are accounted for, apart from the convergence of the Pareto fronts, while comparing them for parity [91]. Moreover, IGD is not affected by outliers as much as it affects GD because it averages the distance between all the points on the reference Pareto front to their corresponding nearest points on the Pareto fronts generated by the metaheuristic algorithms, hence reducing the effect of a single outlier [92,96]. However, both GD and IGD consider only a unidirectional perspective and, are influenced by outliers on P and Q, respectively, to some extent that neither of them can address the worst-case scenario [91]. To address this limitation, Schutze et al. [92] proposed the averaged Hausdorff distance metric that combines the GD and IGD metrics to incorporate a bidirectional perspective. Averaged Hausdorff distance (HD) is calculated as [92]

$$HD(P,Q) = max\left(GD_p(P,Q), IGD_p(Q,P)\right) \tag{19}$$

where larger p values indicate larger penalties for outliers [91]. In this paper, we have chosen the value of p as 1 [91,96]. The algorithm that results in the least value of HD, corresponding to each problem instance, is the one that is capable of finding the nearest optimal solutions to the reference Pareto-optimal front.

5.3.3. Diversity metric

The diversity metric determines the maximum spread of the non-dominated solutions on the Pareto-optimal front. The diversity metric (DM) or Zitzler's M_3^* metric for Pareto fronts comprising only points is calculated as [87,89]

$$M_3^*(K_P) = \sqrt{\sum_{i=1}^{m} \max_{j\in\{1,2,\ldots,|K_P|\}} \max_{k\in K_P\setminus\{k^j\}} ||k^j - k||} \tag{20}$$

where m denotes the number of objectives, K_P denotes the set of points on a Pareto front and $||k^j - k||$ is the distance between two solutions (points) k^j and k on the Pareto-optimal front. We have suitably adapted this metric to calculate the maximum spread of non-dominated solutions on a Pareto-optimal front comprising line segments and points, as shown in Eq. (21).

$$M_3^*(K_L) = \sqrt{\sum_{i=1}^{m} \max_{j\in\{1,2,\ldots,|K_L|\}} \max_{k^l\in K_L\setminus\{k^j\}} \{K^{jl}\}} \tag{21}$$

$$where\ K^{jl} = \begin{cases} \left|\left|k_{1i}^j - k_{2i}^l\right|\right|; & l > j \\ \left|\left|k_{1i}^l - k_{2i}^j\right|\right|; & l < j \end{cases}$$

In Eq. (21), K_L denotes the set of line segments and points on a Pareto front, k^j and k^l represents two non-dominated line segments on K_L, the coordinates of which are represented as $\left(k_{11}^j, k_{12}^j\right)$, $\left(k_{21}^j, k_{22}^j\right)$ and $\left(k_{11}^l, k_{12}^l\right), \left(k_{21}^l, k_{22}^l\right)$, respectively. A point is considered as a line segment with the same end points. The distance K^{jl} between the line segments k^j and k^l is measured as the distance between the first endpoint of one of the line segments and the second endpoint of the other. The M_3^* metric in effect measures the difference in the objective values of the extremal solutions on K_L, and hence the maximum spread [87]. A higher value of DM indicates that the non-dominated solutions on the Pareto-optimal front are more diverse. Therefore, the metaheuristic algorithm that achieves the

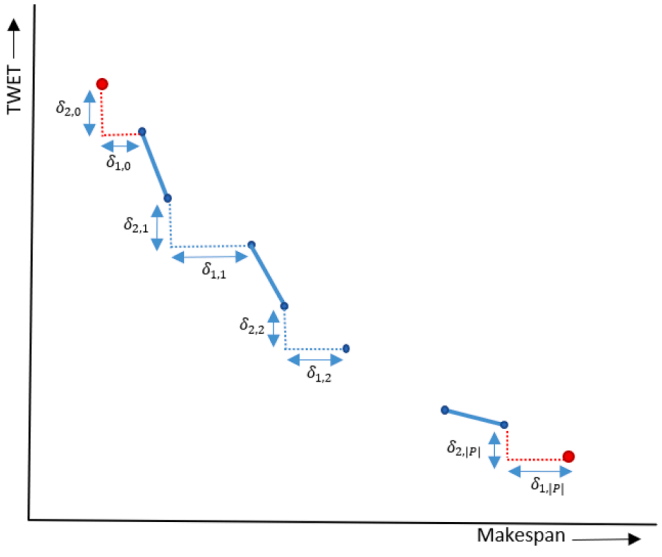


Fig. 5. Calculation of Δ metric for a Pareto-optimal front comprising line segments and points.

highest value of DM is considered the best concerning solution diversity.

5.3.4. Distribution (Δ) metric

The **Δ** metric measures how uniformly the non-dominated solutions are spread on the Pareto-optimal fronts [97]. The **Δ** metric is calculated as [87]

$$\Delta = \max_{j\in\{1,\ldots,m\}} \frac{\left(\delta_{j,0} + \delta_{j,|P|} + \sum_{i=1}^{|P|-1}\left|\delta_{j,i} - \overline{\delta_j}\right|\right)}{\left(\delta_{j,0} + \delta_{j,|P|} + (|P|-1)\overline{\delta_j}\right)} \tag{22}$$

where m denotes the number of objectives, $|P|$ denotes the number of non-dominated solutions on the Pareto optimal front, $\overline{\delta_j}\ \forall j = 1,\ldots,m$ denotes the average of all the deviations in the j^{th} objective $\delta_{j,i}\ \forall i = 1,\ldots, |P|-1$ between consecutive points on the Pareto-optimal front and $\delta_{j,0}$ and $\delta_{j,|P|}$ denote the deviations in the j^{th} objective between the reference points and the solution points at the corresponding extreme ends of the Pareto optimal front, as shown in Fig. 5. We have suitably adapted this metric to measure uniformity in the distribution of non-dominated solutions on Pareto-optimal fronts comprising line segments and points. In Fig. 5, $\delta_{1,i}$ and $\delta_{2,i}\ \forall i = 1,\ldots,|P|-1$ denotes the deviations in the makespan and TWET objectives, respectively, between consecutive line segments or points on the Pareto-optimal front. The Δ metric for the Pareto-optimal front comprising line segments and points is then calculated by substituting the values of these deviations in Eq. (22).

A smaller value of Δ indicates that the non-dominated solutions are uniformly distributed on the Pareto-optimal front [87]. Hence, the algorithm that achieves the lowest value of Δ provides the most uniformly distributed solutions on the Pareto-optimal front.

5.4. Performance comparison

Tables 2 and 3 show the average values of the metrics obtained for the Pareto-optimal fronts generated in ten runs for each of the 24 problem instances using the metaheuristic algorithms. Since the performance metrics obtained with each metaheuristic algorithm are large, we have reported the percentage deviations of the average metric values from the best average metric value obtained with all the employed algorithms for each problem instance. The percentage deviation of a metric M, denoted by $M(\%)$ for ten runs of a problem instance using an algorithm k is calculated as

$$M(\%) = \left(\frac{M_{avg}^{best} - M_{avg}^{k}}{M_{avg}^{best}}\right)\text{x}100 \tag{23}$$

where M_{avg}^{k} denotes the average value of the metric M obtained for ten runs of the algorithm k, and M_{avg}^{best} denotes the best average value of the metric M achieved among all the algorithms. A percentage deviation of metric M close to 0 obtained by algorithm k indicates that algorithm k is the most efficient among all the employed metaheuristic algorithms corresponding to the metric M. The more the positive or negative percentage deviations in metric M from 0, the inferior is the algorithm k corresponding to metric M.

The performance of the neighbourhood search-based methodologies presented in this paper has also been compared with that of a hybrid multi-objective particle swarm optimization and Genetic algorithm (MOPSO-GA) adopted from the literature [39]. The MOPSO-GA algorithm, which is a population-based metaheuristic algorithm, does not require a crowding distance or a region-based selection technique as part of its procedure. The methodology begins with an initial population subjected to a fitness evaluation. A global best archive, *g-best* stores the solutions with the best fitness, eliminating the dominated solutions in the archive. In the process, whenever the size of the archive exceeds its limit, the ε-dominated solutions are eliminated from the archive. Each solution also has a personal best archive *p-best* updated by a similar procedure. Each solution in the population is updated with a randomly

Table 2
Comparison of hypervolume and averaged Hausdorff distance obtained with the metaheuristic algorithms.

Metrics			Hypervolume (% deviation from the best average value)							Averaged Hausdorff Distance(% deviation from the best average value)						
Sl.No.	No.of jobs	Problem instance	MOVND	MOIVND	MOVND_PR	MOVNS	MOVNS_I	MOILS	MOPSO-GA	MOVND	MOIVND	MOVND_PR	MOVNS	MOVNS_I	MOILS	MOPSO-GA
1	20	P20_T0.1_R0.8	1.315	0	0.168	1.586	0.943	1.043	4.947	−862.6	0	−23.7	−771.8	−672.0	−724.4	−553.5
2	20	P20_T0.1_R1.2	1.659	0	0.013	1.624	1.074	1.807	5.417	−446.5	0	−0.6	−394.1	−212.5	−390.1	−651.9
3	20	P20_T0.3_R0.8	4.385	0	0.011	4.769	3.546	3.260	12.220	−1005.9	0	−7.6	−897.6	−780.2	−522.5	−871.9
4	20	P20_T0.3_R1.2	2.687	0	0.127	3.309	2.614	2.354	12.247	−108.8	0	−12.5	−45.7	−108.0	−77.8	−330.2
5	30	P30_T0.1_R0.8	1.811	0	0.213	1.850	1.979	4.728	8.880	−122.8	0	−30.6	−99.2	−254.8	−313.5	−512.8
6	30	P30_T0.1_R1.2	1.894	0	0.028	1.151	1.585	3.678	5.483	−218.6	0	−11.9	−91.3	−199.0	−584.8	−454.6
7	30	P30_T0.3_R0.8	3.326	0	0.742	4.403	4.008	6.576	19.765	−190.8	0	−70.2	−177.5	−91.6	−129.2	−258.9
8	30	P30_T0.3_R1.2	2.699	0	0.317	2.110	1.516	5.518	10.366	−115.5	−10.8	0	−145.3	−64.3	−238.9	−219.7
9	40	P40_T0.1_R0.8	1.934	0	0.277	1.727	1.569	5.606	9.073	−140.5	−21.8	0	−252.7	−229.8	−694.0	−305.1
10	40	P40_T0.1_R1.2	1.875	0	0.176	1.515	1.929	6.611	8.284	−182.1	0	−10.9	−124.6	−243.8	−357.7	−191.6
11	40	P40_T0.3_R0.8	3.664	0	0.710	5.311	4.465	10.256	17.488	−456.2	0	−25.9	−288.5	−200.5	−668.1	−392.9
12	40	P40_T0.3_R1.2	4.242	0	0.361	3.761	6.734	10.788	18.653	−223.1	0	−2.7	−158.9	−244.3	−331.2	−392.4
13	50	P50_T0.1_R0.8	4.011	0	0.107	3.546	3.911	12.768	14.568	−324.3	−24.6	0	−290.7	−173.0	−686.2	−307.3
14	50	P50_T0.1_R1.2	1.761	0	0.192	1.686	0.000	8.472	9.852	−212.2	0	−72.3	−452.0	−313.5	−939.9	−605.8
15	50	P50_T0.3_R0.8	7.978	0	0.768	8.437	7.990	18.675	34.715	−520.2	0	−70.1	−314.2	−352.9	−640.7	−405.3
16	50	P50_T0.3_R1.2	4.127	0	0.095	3.584	4.123	12.668	17.507	−284.1	−61.8	0	−120.1	−265.5	−894.6	−846.0
17	75	P75_T0.1_R0.8	1.861	0	0.110	2.341	1.995	9.562	11.797	−105.3	0	−9.9	−213.4	−204.2	−620.9	−612.8
18	75	P75_T0.1_R1.2	1.986	0	0.089	1.945	2.075	10.856	10.870	−46.9	0	−32.1	−217.6	−193.3	−1197.5	−707.3
19	75	P75_T0.3_R0.8	4.408	0	0.283	4.096	4.122	14.293	19.895	−220.7	0	−23.0	−193.4	−362.4	−1029.1	−479.2
20	75	P75_T0.3_R1.2	4.562	0.092	0	4.163	4.217	16.085	20.456	−299.8	−32.3	0	−298.3	−329.0	−1231.4	−1141.3
21	100	P100_T0.1_R0.8	1.857	0	0.029	2.938	2.833	14.445	14.578	−174.6	0	−8.9	−145.1	−179.7	−711.1	−690.9
22	100	P100_T0.1_R1.2	1.449	0.038	0	2.032	2.505	12.956	12.420	−21.5	0	−41.0	−110.2	−212.5	−724.0	−341.5
23	100	P100_T0.3_R0.8	4.268	0	0.072	5.140	4.576	21.437	26.418	−143.6	0	−40.1	−136.5	−229.8	−422.7	−214.7
24	100	P100_T0.3_R1.2	3.607	0.058	0	4.061	4.255	17.545	21.809	−85.9	−16.2	0	−170.7	−209.9	−950.2	−885.1
Average			3.057	0.008	0.204	3.212	3.107	9.666	14.488	−271.3	−7.0	−20.6	−254.6	−263.6	−628.4	−515.5

selected solution from its *p-best* archive and the *g-best* archive using the crossover and mutation operations. The new solutions generated are used to update the *g-best* and *p-best* archives. This process of randomly selecting solutions from the *g-best* and *p-best* archives, subjecting them to crossover and mutation, and using the new solutions to update the archives is repeated until the stopping criterion (*CPU_TimeLimit*) is met. Tables 2 and 3 show the results obtained with the MOPSO-GA algorithm alongside the neighbourhood search metaheuristics presented in this paper.

Table 2 shows the percentage deviation of the hypervolume and the averaged Hausdorff distance values obtained with the metaheuristic algorithms. For all the problem instances with up to 50 jobs and most of the problem instances with 75 and 100 jobs, the MOIVND algorithm has the lowest percentage deviation from the best average hypervolume achieved among all the algorithms. This indicates that the MOIVND algorithm has the best convergence compared to the other metaheuristic algorithms presented in this paper. Table 2 also shows that for most problem instances, the MOIVND algorithm has the least percentage deviation from the best averaged Hausdorff distance achieved among all the algorithms. Hence, among the employed metaheuristic algorithms, MOIVND is the algorithm capable of finding the Pareto-optimal solutions closest to the non-dominated solutions on the reference Pareto-optimal front.

Table 3 shows the percentage deviations of the diversity metric and the distribution metric values obtained with the metaheuristic algorithms. The values obtained by the algorithms for the diversity metric for most problem instances indicate that the MOIVND algorithm has the least percentage deviation from the best average diversity metric value achieved among all the algorithms. This reveals that the MOIVND algorithm performs better than all the other metaheuristic algorithms presented in providing the most diverse solutions. Table 3 also shows the percentage deviations of the distribution metric obtained with the employed metaheuristic algorithms. It can be observed that the MOIVND algorithm has a large percentage deviation from the best average value of the distribution metric obtained among all the algorithms. This indicates that the solutions provided by the MOIVND algorithm are not uniformly spaced on the Pareto-optimal front. However, the percentage deviations in the hypervolume, the averaged Hausdorff distance and the diversity metric achieved by the MOIVND algorithm indicate that the MOIVND algorithm is superior to the other algorithms.

Fig. 6 shows the Pareto-optimal fronts comprising non-dominated line segments and points obtained by each of the metaheuristic algorithms with different problem instances of 40, 50, 75 and 100 jobs. The graphs reveal that the MOPSO-GA algorithm performs far inferior to the other algorithms. The graphs also reveal that the distribution and spacing of the non-dominated solutions obtained with all the algorithms are uniform and smaller in the lower part of the Pareto-optimal front with better TWET values. The spacing and distribution become larger and coarser in the upper part of the Pareto-optimal front with better makespan values. The graphs also reveal that all the algorithms show a similar level of convergence in the lower part of the Pareto-optimal front. The upper part of the Pareto-optimal fronts shows that the MOIVND and MOVND_PR algorithms perform relatively better in solution quality. Since optimizing the makespan objective requires higher exploration in the sequences of jobs compared to the TWET objective, which was discussed in Section 5.2, the perturbation phase in the MOIVND algorithm can be attributed to its better performance in the upper part of the Pareto-optimal front, which allows it to escape the local minima and subsequently find improved solutions in the succeeding improvement phase. The influence of the parameters, *perturb_iter* and *seq_wl*, used in the MOIVND algorithm is presented in Section 5.6.

5.5. Non-parametric statistical analysis

Non-parametric statistical tests are statistical methods used to

Table 3
Comparison of diversity metric and distribution metric obtained with the metaheuristic algorithms.

Metrics			Diversity metric (% deviation from the best average value)							Distribution metric (% deviation from the best average value)						
Sl.No.	No.of jobs	Problem instance	MOVND	MOIVND	MOVND_PR	MOVNS	MOVNS_I	MOILS	MOPSO-GA	MOVND	MOIVND	MOVND_PR	MOVNS	MOVNS_I	MOILS	MOPSO-GA
1	20	P20_T0.1_R0.8	27.463	0	2.027	26.047	23.928	25.002	21.377	−0.059	−2.776	−3.084	0	−0.434	−0.307	−1.292
2	20	P20_T0.1_R1.2	20.503	0	0.000	19.547	12.906	19.066	24.883	0	−1.574	−1.493	−0.336	−0.901	−0.345	−0.304
3	20	P20_T0.3_R0.8	23.391	0	0.611	21.072	19.084	13.306	20.595	−0.305	0	−0.001	−0.272	−0.305	−0.279	−1.123
4	20	P20_T0.3_R1.2	29.686	0	2.505	20.576	34.724	24.040	36.799	−1.369	0	−0.243	−0.954	−0.568	−0.443	−2.359
5	30	P30_T0.1_R0.8	15.090	0	3.486	14.629	18.541	20.037	25.592	−1.318	−4.321	−2.885	−1.526	−0.362	−1.554	0
6	30	P30_T0.1_R1.2	4.467	3.911	3.911	0.665	1.017	7.371	0	−1.550	0	−0.334	−2.888	−3.265	−1.118	−5.013
7	30	P30_T0.3_R0.8	26.217	1.376	0	27.380	21.309	24.678	28.307	−0.254	−4.419	−6.597	−0.055	−0.102	0	−0.557
8	30	P30_T0.3_R1.2	6.021	1.336	0	8.661	4.512	13.328	12.285	−2.086	−0.930	−1.082	0	−1.464	−0.667	−0.105
9	40	P40_T0.1_R0.8	5.418	0	1.104	5.826	6.297	11.770	7.956	−1.673	−2.821	−2.335	−1.150	−1.273	0	−1.885
10	40	P40_T0.1_R1.2	11.554	1.507	0	7.753	11.422	13.817	8.310	0	−4.594	−4.944	−1.515	−0.215	−0.323	−4.853
11	40	P40_T0.3_R0.8	20.704	5.894	0	15.016	16.139	26.406	18.619	−0.985	−3.484	−5.497	−3.765	−1.711	0	−3.225
12	40	P40_T0.3_R1.2	13.039	0	0.263	9.363	13.899	17.226	21.373	−0.570	−3.380	−3.584	−1.122	−0.401	−1.494	0
13	50	P50_T0.1_R0.8	10.307	0	2.453	9.494	6.405	14.284	12.838	−0.611	−3.975	−2.301	−0.800	−2.901	0	−0.721
14	50	P50_T0.1_R1.2	4.467	0	0.537	8.626	4.057	14.094	12.168	−4.175	−5.752	−5.241	−1.109	−5.708	−0.671	0
15	50	P50_T0.3_R0.8	32.126	0	9.535	23.885	25.118	37.089	33.361	0	−6.820	−3.966	−1.819	−1.482	−2.462	−4.344
16	50	P50_T0.3_R1.2	13.250	0	4.331	7.821	10.847	23.054	25.446	−1.857	−3.959	0	−2.185	−3.254	−1.447	−0.759
17	75	P75_T0.1_R0.8	4.349	0	2.526	6.939	6.916	14.010	16.712	−5.257	−6.386	−5.357	−3.257	−4.582	−3.593	0
18	75	P75_T0.1_R1.2	1.531	0	0.718	7.022	6.171	14.703	14.073	−3.548	−4.232	−3.390	−1.645	−1.890	−0.155	0
19	75	P75_T0.3_R0.8	10.213	0	2.375	13.471	15.159	23.908	20.161	−4.836	−5.661	−3.447	−0.578	−3.598	−3.851	0
20	75	P75_T0.3_R1.2	10.678	0.810	0	7.514	9.526	27.794	29.655	−2.467	−2.212	−1.209	−1.875	−2.096	−3.186	0
21	100	P100_T0.1_R0.8	4.937	0	0.446	8.005	6.407	13.262	18.638	−5.317	−7.074	−6.429	−3.362	−4.420	−4.221	0
22	100	P100_T0.1_R1.2	29.715	29.479	30.114	32.828	34.941	0	37.483	−5.060	−4.702	−3.878	−3.045	−2.589	0	−0.454
23	100	P100_T0.3_R0.8	13.019	0	5.080	15.343	19.457	31.412	24.789	−6.913	−9.593	−6.992	−3.421	−3.981	−5.811	0
24	100	P100_T0.3_R1.2	3.014	2.854	0	8.736	6.883	25.588	27.619	−6.106	−5.516	−4.879	−5.369	−5.362	−7.769	0
Average			14.215	1.965	3.001	13.593	13.986	18.969	20.793	−2.346	−3.924	−3.299	−1.752	−2.203	−1.654	−1.125

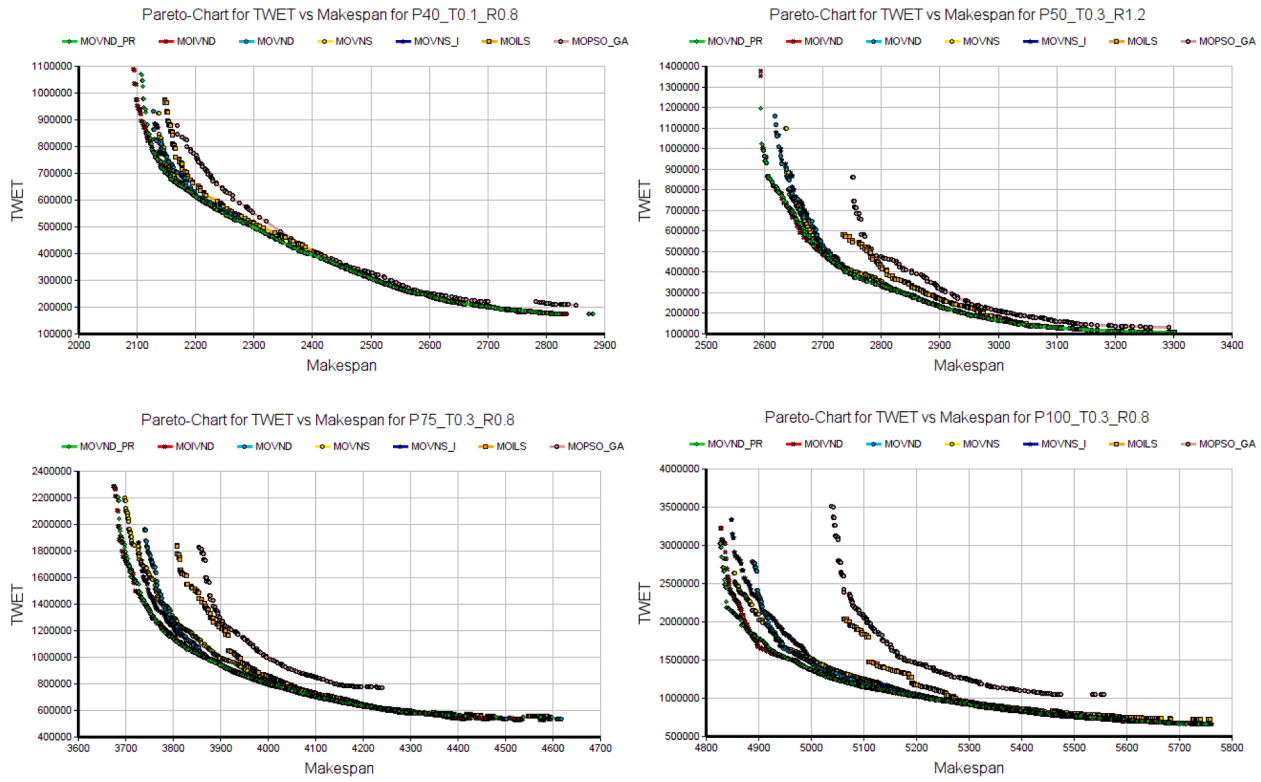


Fig. 6. Pareto-optimal fronts generated by the presented metaheuristic algorithms for four problem instances of different sizes.

analyze data that do not have a specific distribution [98]. To enhance the interpretability of the performance metrics obtained for the metaheuristic algorithms presented in this paper, we have subjected the metric data to a widely used non-parametric statistical test known as the Friedman test. The test has been performed on the hypervolume and the averaged Hausdorff distance metrics to analyze whether the performance of the metaheuristic algorithms differs significantly from each other in terms of the two metrics [99]. The Friedman test evaluates the null hypothesis (H_0) that there are no significant differences between the performance of the algorithms. The alternate hypothesis (H_a) is that at least one of the algorithms performs significantly differently from the others. A statistical measure p-value, calculated based on the ranks obtained by the algorithms in the tests, is used to interpret whether significant differences exist in the performance of the algorithms [98]. A p-value less than the significance level α (typically 0.05) rejects the null hypothesis, implying that significant differences exist between the performance of the algorithms. A significance level of 0.05 implies a 5 % risk of concluding that there are significant differences between the performance of the algorithms when there are none [100].

Table 4
Ranks, statistics and related p-values achieved by the Friedman test in terms of the hypervolume and averaged Hausdorff distance metrics.

Metaheuristic algorithms	Average Friedman test rankings for the algorithms	
	Hypervolume metric	Averaged Hausdorff distance metric
MOVND	4.0833	4.5416
MOIVND	1.1249	1.2500
MOVND_PR	1.8749	1.7916
MOVNS	4.1250	4.0000
MOVNS_I	4.1249	4.2916
MOILS	5.7083	6.2916
MOPSO-GA	6.9583	5.8333
Statistic	125.9464	110.2142
p-value	5.62E-11	9.94E-11

Table 4 shows the average ranks of the metaheuristic algorithms obtained for multiple problem instances corresponding to the hypervolume and the averaged Hausdorff distance metrics and the statistics and p-values obtained by the Friedman test, computed with 5 degrees of freedom, using the CONTROLTEST package available for download at https://sci2s.ugr.es/sicidm [98]. Table 4 shows that the MOIVND algorithm has obtained the lowest rank among all the metaheuristic algorithms in terms of both the hypervolume and averaged Hausdorff distance metrics, indicating that the MOIVND algorithm is the best-performing algorithm followed by the MOVND_PR algorithm. Further, the p-values obtained corresponding to both metrics are less than 0.05, indicating strong evidence against the null hypothesis. This implies that at least one of the metaheuristic algorithms presented in this paper has performed significantly better than the others. Further, to control the risk of obtaining significant results due to random odds, we performed several post-hoc tests, namely the Holm, Hommel, Rom, Finner and Li tests, to adjust the p-values obtained by the initial Friedman test, considering the MOIVND algorithm as the control method [98]. The best-performing algorithm identified by the Friedman test is known as the control method. The unadjusted p-values obtained by the Friedman test and the corresponding adjusted p-values obtained with the post-hoc test procedures, corresponding to the hypervolume and averaged Hausdorff distance metrics, are shown in Tables 5 and 6, respectively.

Tables 5 and 6 show that the unadjusted p-value obtained corresponding to the hypervolume and averaged Hausdorff distance metric for all the algorithms except the MOVND_PR algorithm is less than 0.05. This indicates that the MOIVND algorithm performs significantly better than all the algorithms presented in this paper except the MOVND_PR algorithm. The adjusted p-values obtained by the post-hoc procedures denoted by p_{Holm}, p_{Homm}, p_{Rom}, p_{Finn}, and p_{Li}, respectively, in Tables 5 and 6 also imply that the performance of the MOIVND algorithm is not significantly different from the MOVND_PR algorithm but is significantly better than the remaining algorithms. Therefore, the statistical analysis performed on the metric data obtained for the algorithms

Table 5
Adjusted p-values for 1xN comparisons among all the algorithms in terms of hypervolume (MOIVND is the control method).

Algorithms	Unadjusted p	p_{Holm}	p_{Homm}	p_{Rom}	p_{Finn}	p_{Li}
MOVND	2.096E-6	6.016E-6	4.193E-6	4.193E-6	3.008E-6	2.719E-6
MOVND_PR	0.2291	0.2291	0.2291	0.2291	0.2291	0.2291
MOVNS	1.504E-6	6.016E-6	3.144E-6	4.193E-6	3.008E-6	1.950E-6
MOVNS_I	1.504E-6	6.016E-6	3.144E-6	4.193E-6	3.008E-6	1.950E-6
MOILS	1.986E-13	9.9337E-13	9.933E-13	9.446E-13	5.961E-13	2.577E-13
MOPSO - GA	8.427E-21	5.056E-20	5.056E-20	4.8080	0.0000	1.093E-20

Table 6
Adjusted p-values for 1xN comparisons among all the algorithms in terms of averaged Hausdorff distance (MOIVND is the control method).

Algorithms	Unadjusted p	p_{Holm}	p_{Homm}	p_{Rom}	p_{Finn}	p_{Li}
MOVND	1.303E-7	5.2123E-7	5.212E-7	4.970E-7	2.606E-7	2.119E-7
MOVND_PR	0.3850	0.3850	0.3850	0.3850	0.3850	0.3850
MOVNS	1.034E-5	2.069E-5	2.069E-5	2.069E-5	1.241E-5	1.682E-5
MOVNS_I	1.074E-6	3.222E-6	3.222E-6	3.222E-6	1.611E-6	1.747E-6
MOILS	6.234E-16	3.740E-15	3.740E-15	3.556E-15	3.996E-15	1.013E-15
MOPSO - GA	1.986E-13	19.933E-13	9.933E-13	9.4469E-13	5.9618E-13	3.230E-13

undoubtedly indicates that the MOIVND algorithm is the best-performing algorithm, followed by the MOVND_PR algorithm with a significance level of 0.05. Further, the p-values obtained for the MOILS and MOPSO-GA algorithms corresponding to both the performance metrics are relatively very much below the significance level, which implies that the MOILS and MOPSO-GA algorithms are the worst-performing algorithms among the metaheuristic algorithms presented in this paper.

5.6. Influence of the parameters used in MOIVND

Fig. 7 shows the influence of the parameters k_1 and k_2 in the initial solution generation on the performance of the MOIVND algorithm for the problem instances P50_T0.1_R0.8 and P75_T0.1_R0.8 with 50 and 75 jobs, respectively. Various fixed pairs of values for k_1 and k_2, as well as random values of k_1 and k_2 generated within different value ranges, were experimented with, as represented on the x-axis of the graphs in the figure. For instance, F1,5 on the x-axis denotes that the constant values 1 and 5 were assigned to k_1 and k_2, respectively. Similarly, (R1, R2) on the x-axis denotes that k_1 and k_2 were randomly generated over the value ranges denoted by R1 and R2, representing the ranges 1 to 5 and 1 to 10, respectively. The best (BEST) and average (AVG) values of the ten runs obtained for the hypervolume metric for each pair of k_1 and k_2 are shown in the graphs. The graphs reveal that the BEST and AVG values of hypervolume are relatively higher when both k_1 and k_2 are randomly generated in the range R2, for both problem instances. Relatively closer values of hypervolume are also obtained when k_1 and k_2 are assigned fixed values of 10 and 5, respectively, as shown in the graphs. Generating k_1 and k_2 over a range of values is preferred over fixed values since a multi-objective optimization problem requires experimenting with all possible combinations of input sequences. Therefore, selecting a suitable range of values in which the two parameters of the initial population generation are randomly generated is significant in the performance of the MOIVND algorithm.

Fig. 8 shows the influence of the perturbation parameter, *perturb_iter*, on the performance of the MOIVND algorithm for the problem instances P50_T0.1_R0.8 and P75_T0.1_R0.8 with 50 and 75 jobs, respectively. The parameter *perturb_iter* is varied from 0–40 in steps of 5, as shown in the figure. For *perturb_iter*=0, the perturbation stage was not performed, and the Pareto-optimal front was generated using only one iteration of the improvement phase. The graphs reveal that the BEST and AVG values of the HV metric are relatively higher in the range 5–10 of *perturb_iter*. This indicates that as the value of *perturb_iter* is increased from 0, the higher perturbation rate allows the algorithm to explore the solution space more and find better solutions. However, as the value of *perturb_iter* is increased beyond 10, the higher perturbation rate does not yield good results as higher exploration leads to significant deviations from the previous best solutions, thus requiring a large number of iterations to converge further in the subsequent improvement phase. This eventually reduces the number of times the solutions are perturbed and improved within the set *CPU_TimeLimit*. Therefore, the influence of the parameter *perturb_iter* is reasonably significant in the performance of the MOIVND algorithm.

Fig. 9 shows the influence of the parameter *seq_wl* on the

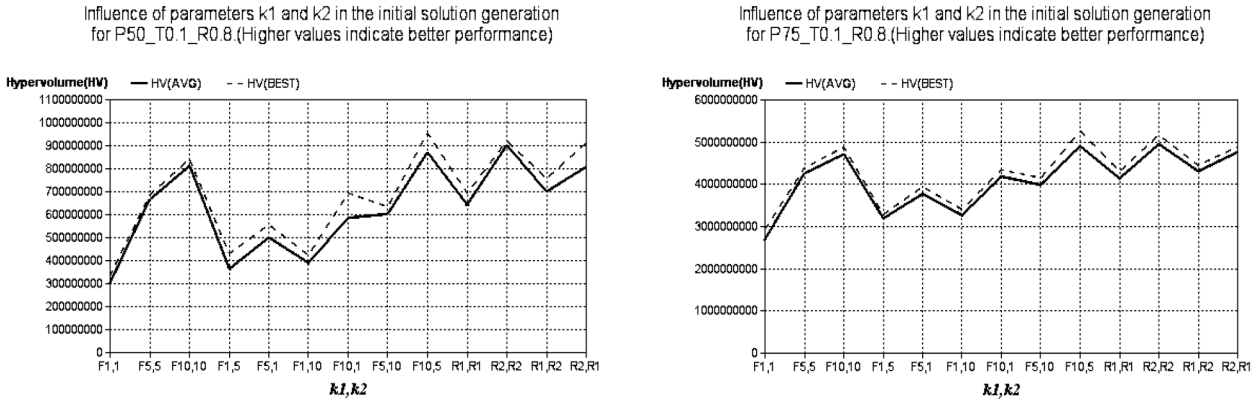


Fig. 7. Influence of the parameters k_1 and k_2 in the initial solution generation of the MOIVND algorithm.

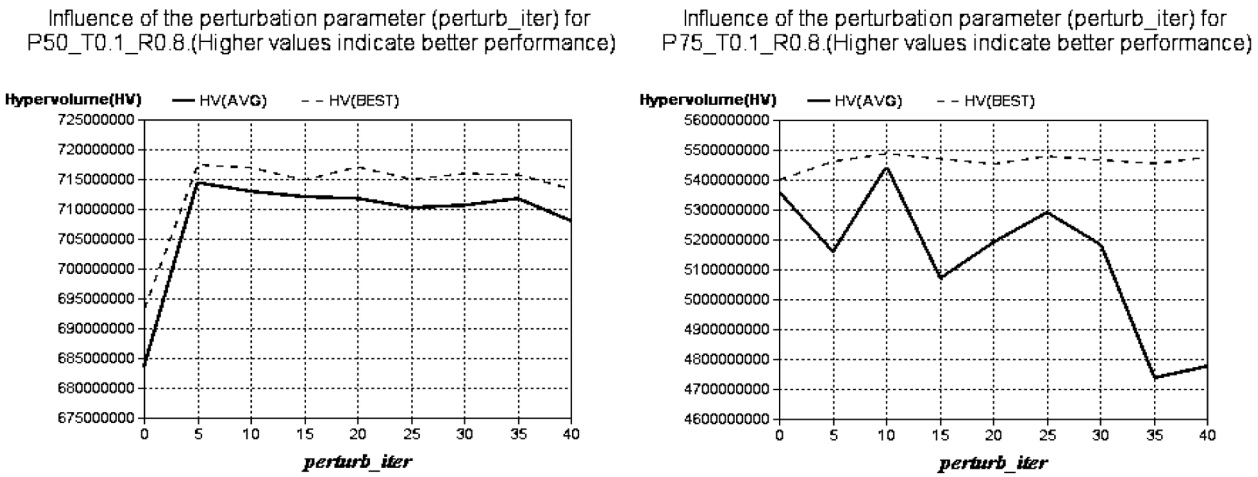


Fig. 8. Influence of the perturbation parameter on the performance of the MOIVND algorithm.

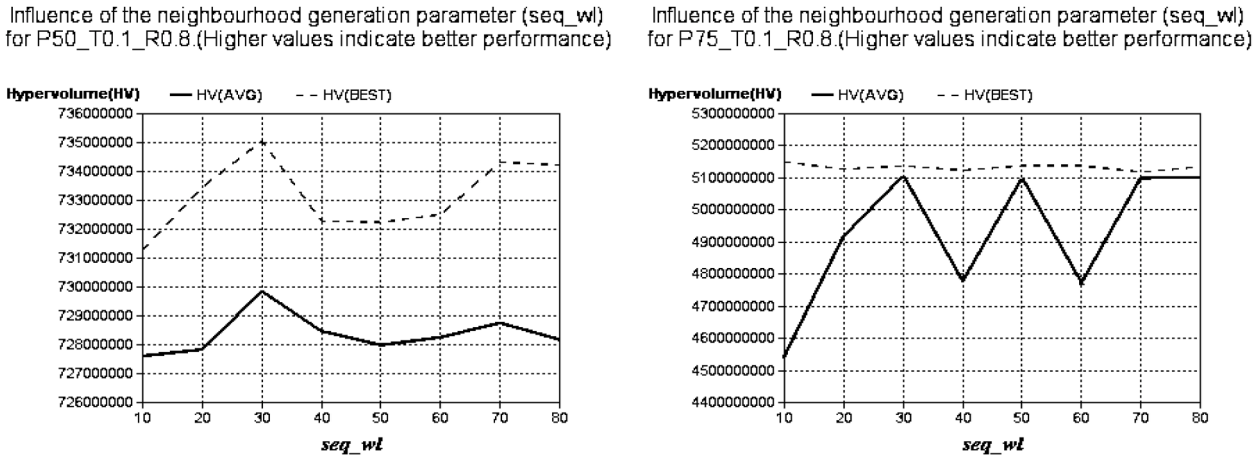


Fig. 9. Influence of the neighbourhood generation parameter *seq_wl* on the performance of the MOIVND algorithm.

performance of MOIVND for the problem instances P50_T0.1_R0.8 and P75_T0.1_R0.8 with 50 and 75 jobs, respectively. As discussed in Section 5.2, each time the neighbourhoods were generated using a neighbourhood structure in the MOIVND algorithm, the first *seq_wl* number of sequences of jobs with the best makespan values in the Pareto-optimal front were subjected to neighbourhood generation without imposing the limits, *swap_limit, shift_limit, dshift_limit* and *tshift_limit*. The parameter *seq_wl* is varied in the range 10–80 in steps of 10, as shown in the figure. The graph corresponding to the instance P50_T0.1_R0.8 reveals that the AVG and BEST values of the HV metric corresponding to *seq_wl*=30 are

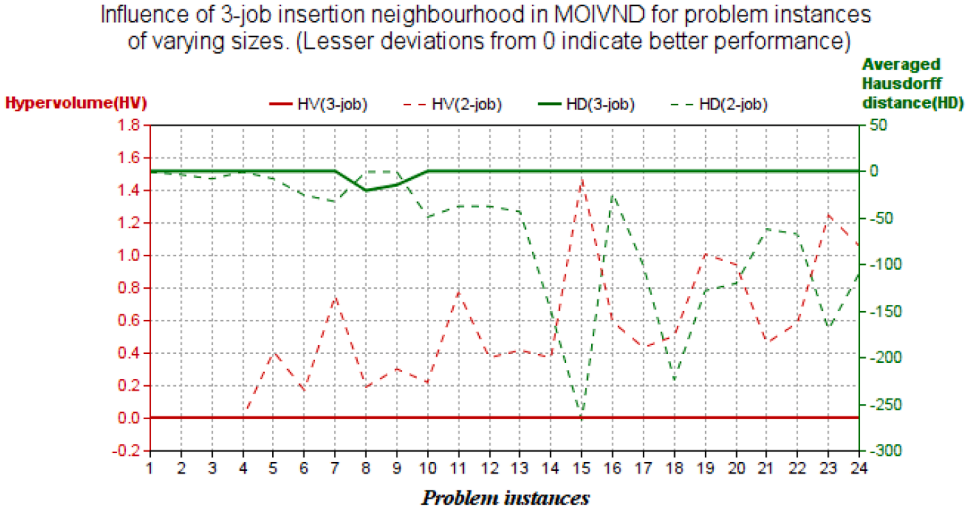


Fig. 10. Influence of 3-job insertion neighbourhood on the performance of the MOIVND algorithm.

the best among all the other values of *seq_wl,* while the graph corresponding to the instance P75_T0.1_R0.8 shows an increasing trend in the AVG value of HV metric up to *seq_wl*=30 and random fluctuations beyond it. The graphs also reveal that the performance deteriorates relatively as *seq_wl* is increased beyond 30. This indicates that the generation of neighbourhoods beyond the limits for 30 or more sequences of jobs in a Pareto-optimal front does not improve the solutions considerably, as generating a large number of neighbourhoods increases the computation time required to generate the TWET-makespan trade-off curves and their Pareto-optimal fronts. This eventually leads to a decrease in the number of times the perturbations are performed within the set *CPU_TimeLimit*. Therefore, the influence of the parameter *seq_wl* is also significant in the performance of the MOIVND algorithm.

Fig. 10 shows the influence of the 3-job insertion neighbourhood generation mechanism on the performance of the MOIVND algorithm for problem instances of varying sizes. The percentage deviations in the hypervolume and averaged Hausdorff distance metrics, obtained by the MOIVND algorithms utilising the four neighbourhood structures $\{N_1,N_2,N_3,N_4\}$ and excluding the 3-job insertion neighbourhood structure (i.e., utilising $\{N_1,N_2,N_3\}$) respectively for each problem instance, calculated using Eq. (23) are shown as two separate graphs. The graph on the left shows the percentage deviations from the best hypervolume (HV) values, and the graph on the right shows the percentage deviations from the best averaged Hausdorff distance (HD) values obtained by the MOIVND algorithms utilising up to 2-job and up to 3-job insertion neighbourhoods depicted using the dashed and solid lines respectively. As mentioned in Section 5.4, any positive or negative deviations from zero indicate that the corresponding algorithm performs inferior to the best-performing algorithm. The graphs reveal that the MOIVND algorithm utilising up to 3-job insertion neighbourhoods has no deviation from zero in the hypervolume value for all the problem instances and no deviation from zero in the averaged Hausdorff distance metric for all except two smaller problem instances. This indicates that the 3-job insertion neighbourhood generation mechanism significantly improves the performance of the MOIVND algorithm. The reason can be attributed to the sequence-dependent setup times considered in the problem, which leads to the clustering of jobs with minimum setup times between them, and multiple jobs insertion neighbourhood structures such as 2-job and 3-job insertions can improve solutions by shifting consecutive jobs in clusters in the sequence. However, experiments with adding a 4-job insertion neighbourhood structure did not yield better results due to increased computation times required in the improvement phase, which reduced the number of iterations of the perturbation phase within the *CPU_TimeLimit*.

5.7. Managerial implications

The rising demand for customer-centric and cost-effective production in industrial systems has enunciated the need for optimization in manufacturing operations. Manufacturers often find it challenging to meet production deadlines while maintaining lower production costs, particularly in just-in-time (JIT) manufacturing environments. The JIT manufacturing systems primarily focus on ensuring the timely delivery of the exact quantities of products while minimizing inventory levels. In several real-world JIT manufacturing systems, a single machine or a single processor often becomes a bottleneck when the demand placed on it exceeds its handling capacity. Such a situation may arise either when a production facility runs on a single machine or when a particular machine in a multi-machine facility is overloaded. Realizing JIT goals in these scenarios entails the simultaneous optimization of several operational performance metrics, which necessitates devising optimal scheduling techniques. The scheduling techniques proposed in this research best suit such scheduling environments with a single machine.

The just-in-time single machine scheduling problem (JIT-SMSP) considered in this paper, which includes the bi-objective optimization of total weighted earliness-tardiness (TWET) and makespan in a scheduling environment with distinct due windows and sequence-dependent setup times (SDST), allowing idle times in the schedule, reflects numerous realistic make-to-order manufacturing environments [101]. Reducing makespan in these applications results in tighter schedules, facilitating the efficient utilization of resources, while the reduction of TWET focuses on meeting deadlines, improving customer satisfaction, and minimizing inventories. The efforts to optimize the two objectives are conflicting in nature since optimizing TWET results in the jobs in the schedule being clustered around their earliest due dates, inserting idle times in the schedule, while optimizing makespan focuses on completing all the jobs in the sequence as early as possible with no idle times in the schedule. This conflicting nature necessitates devising a method of trade-off plot generation between the two objectives for a given sequence of jobs, which is now addressed by the optimal timing algorithm (OTA) proposed in this paper. Though the problem is practically relevant from an industrial application perspective, to the best of our knowledge, no research has been reported in the literature to solve practical size instances due to the nonexistence of suitable methodologies for generating trade-off plots and Pareto fronts comprising line segments and points between the considered objectives. The problem is well-known to be NP-hard, and Pareto-based metaheuristics have been proven to be the most efficient method in the literature for solving such complex problems. The Pareto-based metaheuristic solution approaches proposed in this paper are computationally efficient to generate the Pareto-optimal front in a reasonable computation time. The solutions on the Pareto-optimal front, comprising line segments and points, will allow decision-makers and practitioners to make informed decisions when choosing the optimal sequences and schedules based on their priorities for the two objectives considered in this paper.

Though this research considered scheduling environment with a single machine, JIT manufacturing facilities with several machine environments exist in industries, most of which have been modelled in the literature, viz. parallel machine scheduling, flow shop scheduling, job shop scheduling, assembly shop scheduling, etc. However, each machine scheduling environment will require devising an optimal timing algorithm specific to the problem, and problem-specific operators and mechanisms will need to be developed within the Pareto-based metaheuristic algorithms to solve the problem efficiently. This serves as a motivation to pursue future research in different machine scheduling environments, thereby contributing towards the operational excellence of JIT manufacturing systems.

6. Conclusions

In this paper, three neighbourhood search-based metaheuristic algorithms, namely the multi-objective variable neighbourhood descent (MOVND), multi-objective iterated variable neighbourhood descent (MOIVND) and hybrid multi-objective variable neighbourhood descent - path relinking (MOVND_PR), were proposed to solve the Pareto-based bi-objective optimization of total weighted earliness-tardiness (TWET) and makespan in a single-machine scheduling problem (SMSP) operating in the just-in-time (JIT) environment. The problem environment encompassed scenarios such as distinct job due windows, sequence-dependent setup times between jobs, and idle times allowed to be inserted in the schedules. We proposed an optimal timing algorithm to generate the TWET-makespan piecewise linear trade-off curve for a given sequence of jobs. A Pareto-optimal front generation procedure was adopted from the literature to generate the Pareto-optimal fronts by trimming and merging multiple Pareto fronts comprising line segments and points, where each Pareto front is associated with a single or multiple sequences of jobs. The resulting Pareto-optimal front also comprised line segments and points. The performance of the proposed neighbourhood search-based metaheuristic algorithms was compared with that of three neighbourhood search-based metaheuristic algorithms adopted from the literature, namely multi-objective variable neighbourhood search (MOVNS) algorithm, multi-objective variable

neighbourhood search with intensification (MOVNS_I) algorithm, and multi-objective iterated local search (MOILS) algorithm. The performance of the algorithms was evaluated using a set of 24 problem instances generated using a procedure adopted from the literature with the number of jobs varying from 20 to 100. The metaheuristic algorithms, MOVND, MOIVND, MOVND_PR, MOVNS, MOVNS_I, and MOILS, were run ten times for each problem instance. The Pareto-optimal fronts generated were further analyzed to determine the performance of the algorithms. Since the Pareto-optimal fronts generated by the metaheuristic algorithms comprised line segments and points, we devised four performance measures by suitably adapting similar ones from the literature. The performance evaluation of the metaheuristic algorithms using the performance metrics, namely, hypervolume, averaged Hausdorff distance, and diversity, revealed that the proposed MOIVND algorithm performed better than all the other algorithms, followed by the MOVND_PR algorithm. Considering the overall performance across all the problem instances, the MOIVND algorithm was found to be the best-performing algorithm for generating the best non-dominated solutions on the Pareto-optimal front. The superior performance of the MOIVND algorithm can be attributed to the backtrack perturbation phase, which allows the search to escape from the local optima, and the 3-job insertion neighbourhood structure, which significantly improves its performance compared to that using up to 2-job insertion mechanism. The performance of the neighbourhood search-based methodologies presented in this paper was also compared with that of the population-based hybrid MOPSO-GA algorithm adopted from the literature, and the computational results showed that the proposed MOIVND algorithm, as well as all the other neighbourhood search-based metaheuristic algorithms, performed exceptionally better than the MOPSO-GA algorithm. Further, a statistical analysis performed on the hypervolume and averaged Hausdorff distance metric data confirmed the correctness of the conclusions drawn from the metric data.

To the best of our knowledge, this is the first study that reports metaheuristics to solve a bi-objective JIT machine scheduling problem that generates a Pareto-optimal front comprising line segments and points. A future research direction would be to suitably adapt population-based metaheuristics and compare their performance with the algorithms proposed in this paper. To the best of our knowledge, this is also the first study that reports performance metrics for Pareto-optimal fronts comprising line segments and points in Pareto-based bi-objective scheduling problems. A future research direction would be to devise or adapt other performance metrics to comprehensively analyze the Pareto-optimal fronts generated with different approaches. Another future research direction would be to study the performance of the proposed algorithms for other combinations of bi-objectives in JIT-SMSP, viz., the simultaneous minimization of TWET-total flow time, total earliness-total tardiness, total earliness-makespan, etc. Another future research direction would be to extend the proposed algorithms to other JIT machine scheduling problems, viz. parallel machine scheduling, flow shop scheduling, job shop scheduling, etc., and performing computational studies for the Pareto-based bi-objective optimization of TWET-makespan and other combinations of objectives.

CRediT authorship contribution statement

Sona Babu: Writing – original draft, Visualization, Validation, Software, Methodology, Investigation, Data curation, Conceptualization. **B.S. Girish:** Writing – review & editing, Supervision, Software, Resources, Methodology, Conceptualization.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

The authors thank the Editor and the four Anonymous Reviewers for the insightful comments and suggestions that helped improve this paper.

Appendix A

An illustration of the TWET-makespan trade-off curve generation procedure

A problem instance with $n = 5$ jobs, as shown in Table A.1, is used to illustrate the TWET-makespan trade-off curve generation procedure.

Table A.1
Test instance data for the illustration problem.

Jobs (i)	Processing time (P_i)	Setup Time ($S_{ii'}$) : $i' = 1,2,...n$	Earliest due date (de_i)	Latest due date (dt_i)	Earliness penalty (α_i)	Tardiness penalty (β_i)
1	6	{0,4,3,5,2}	75	80	3	4
2	8	{4,0,5,3,6}	82	85	5	7
3	7	{3,5,0,4,3}	39	45	4	3
4	9	{5,3,4,0,4}	50	59	6	5
5	5	{2,6,3,4,0}	40	44	3	6

Let the given job sequence be $\sigma = \{3,5,4,1,2\}$. The first job $\sigma[1]$=3 is assigned to the machine exactly at its earliest due date (i.e. $C_{\sigma[1]} = 39$) as described in steps 2–5 of Algorithm 1. The position identifier of $\sigma[1]$ is added to the block B as $B = \{1\}$. The corresponding partial schedule $S_1 = \{39\}$ and $TWET(\sigma_1) = 0$. The second job $\sigma[2]$=5 is then assigned contiguously to $\sigma[1]$=3 at $C_{\sigma[2]} = C_{\sigma[1]} + P_{\sigma[2]} + S_{\sigma[1],\sigma[2]} = 39 + 5 + 3 = 47$ since $de_{\sigma[2]} = 40$, as shown in the Gantt chart in Fig. A.1. This corresponds to steps 18–19 of Algorithm 1. The partial schedule $S_2 = \{39, 47\}$ hence obtained has its corresponding $TWET(\sigma_2) = (C_{\sigma[2]} - dt_{\sigma[2]})\beta_{\sigma[2]} = (47 - 44) * 6 = 18$. The position identifiers of the two jobs form a block $B = \{1,2\}$, as shown in step 24 of Algorithm 1, with its cost function slope $SL = \beta_{\sigma[2]} - \alpha_{\sigma[1]} = 6 - 4 = 2$ calculated, as shown in steps 10–14 of Algorithm 2. Since the slope is positive, the jobs with their position identifiers in B are simultaneously left shifted until one of the following cases is encountered, as shown in steps 17–25 of Algorithm 2.

Case 1: The newly scheduled job $\sigma[2]$ is no longer tardy (i.e. at $C_{\sigma[2]} = 44$)
Case 2: A job with its position identifier in B becomes early (i.e. at $C_{\sigma[2]} = 40$)
Case 3: No idle time exists before the job $\sigma[1]$ (i.e. at $C_{\sigma[1]} = P_{\sigma[1]} = 7$)

In this scenario, Case 1 is first encountered on left shifting the jobs with position identifiers in B by 3 units. The job $\sigma[2]$ then becomes no longer tardy with the cost function slope $SL = -\alpha_{\sigma[1]} = -4$, implying that no further left shifting is required. The optimal partial schedule $S_2 = \{36, 44\}$ hence obtained has its corresponding $TWET(\sigma_2) = (de_{\sigma[1]} - C_{\sigma[1]})\alpha_{\sigma[1]} = (39 - 36) * 4 = 12$. Fig. A.1 shows the partial schedule S_2 before and after left shifting.

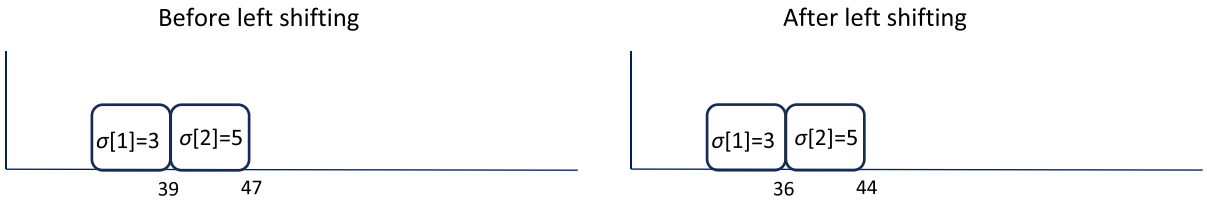


Fig. A.1. Generation of the optimal partial schedule S_2.

The partial schedules S_3 is subsequently generated by assigning the job $\sigma[3]$=4 at $C_{\sigma[3]} = C_{\sigma[2]} + P_{\sigma[3]} + S_{\sigma[2],\sigma[3]} = 44 + 9 + 4 = 57$, which is within its due window [50,59]. This is as shown in steps 18–19 of Algorithm 1. Since the job $\sigma[3]$ is contiguous with $\sigma[2]$, the block B is updated as $B = \{1,2,3\}$. The slope of the cost function corresponding to B remains as $SL = -4$, and no left shifting is done. The optimal partial schedule $S_3 = \{36, 44, 57\}$ and its corresponding $TWET(\sigma_3) = 12$.

Subsequently, the partial schedule S_4 is generated by assigning the job $\sigma[4]$=1 at $C_{\sigma[4]} = de_{\sigma[4]} = 75$ since $C_{\sigma[3]} + P_{\sigma[4]} + S_{\sigma[3],\sigma[4]} = 57 + 6 + 5 = 68$, which is less than $de_{\sigma[4]}$. Since the jobs $\sigma[3]$ and $\sigma[4]$ are non-contiguous with each other, the block B is reset as $B = \{4\}$. This is as shown in steps 9–12 of Algorithm 1. The resulting partial schedule $S_4 = \{36, 44, 57, 75\}$ is optimal with $TWET(\sigma_4) = 12$. The job $\sigma[5]$=2 is then assigned to the machine at $C_{\sigma[5]} = C_{\sigma[4]} + P_{\sigma[5]} + S_{\sigma[4],\sigma[5]} = 75 + 8 + 4 = 87$ contiguously to $\sigma[4]$ since $de_{\sigma[5]} = 82$. The partial schedule $S_5 = \{36, 44, 57, 75, 87\}$ has its corresponding $TWET(\sigma_5) = (de_{\sigma[1]} - C_{\sigma[1]})\alpha_{\sigma[1]} + (C_{\sigma[5]} - dt_{\sigma[5]})\beta_{\sigma[5]} = (39 - 36) * 4 + (87 - 85) * 7 = 26$, and the two contiguous jobs form the block $B = \{4, 5\}$ with its cost function slope $SL = \beta_{\sigma[5]} - \alpha_{\sigma[4]} = 7 - 3 = 4$. Since the slope is positive, the jobs with their position identifiers in B are simultaneously left shifted until one of the following cases is encountered.

Case 1: $\sigma[5]$ is no longer tardy (i.e. at $C_{\sigma[5]} = 85$)
Case 2: A job with its position identifier in B becomes early (i.e. at $C_{\sigma[5]} = 82$)
Case 3: $\sigma[4]$ becomes contiguous with $\sigma[3]$ (i.e. at $C_{\sigma[4]} = C_{\sigma[3]} + P_{\sigma[4]} + S_{\sigma[3],\sigma[4]} = 68$)

In this scenario, Case 1 is first encountered on left shifting the jobs with position identifiers in B by 2 units, generating the optimal partial schedule $S_5 = \{36, 44, 57, 73, 85\}$ with $TWET(\sigma_5) = (de_{\sigma[1]} - C_{\sigma[1]})\alpha_{\sigma[1]} + (de_{\sigma[4]} - C_{\sigma[4]})\alpha_{\sigma[4]} = (39 - 36) * 4 + (75 - 73) * 3 = 18$. Fig. A.2 shows the partial schedule S_5 before and after left shifting.

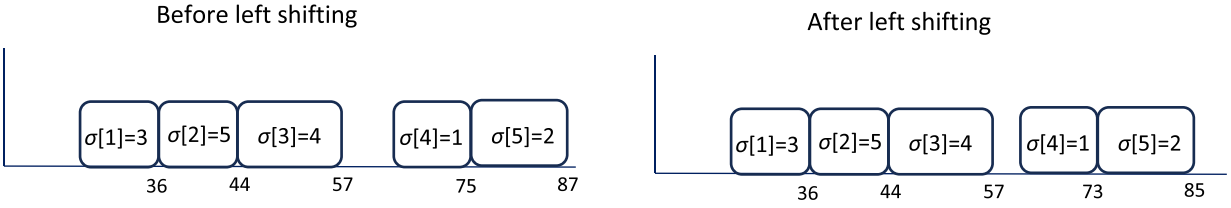


Fig. A.2. Generation of the optimal partial schedule S_5.

Since the cost function slope $SL = -\alpha_{\sigma[4]} = -3$ corresponding to the block $B = \{4, 5\}$, the partial schedule $S_5 = \{36, 44, 57, 73, 85\}$ with $TWET(\sigma_5) = 18$ is optimal. Since all the jobs in σ have been assigned completion times, the partial schedule S_5 will be the optimal TWET schedule S for the given σ. At this stage, the makespan and TWET objective values are stored as the first trade-off point $(M_1, G_1) = (85, 18)$ in the TWET-makespan trade-off plot, as shown in steps 29–32 of Algorithm 1 and 26–28 of Algorithm 2.

Subsequently, to optimize makespan, the last job $\sigma[5]$ in the TWET-optimal schedule and its preceding contiguous job $\sigma[4]$ that form the block $B = \{4, 5\}$ are left shifted by the smallest unit of left shift that leads to encountering one of the following cases as shown in steps 16 and 25 of Algorithm 1.

Case 1: A job with its position identifier in B becomes early (i.e. at $C_{\sigma[5]} = 82$)
Case 2: $\sigma[4]$ becomes contiguous with $\sigma[3]$ (i.e. at $C_{\sigma[4]} = 68$)

In this scenario, Case 1 is the condition encountered by the smallest unit of left shifting, and therefore, the jobs with position identifiers in B are left shifted by 3 units. The schedule thus obtained $S = \{36, 44, 57, 70, 82\}$ with $TWET(\sigma) = (de_{\sigma[1]} - C_{\sigma[1]})\alpha_{\sigma[1]} + (de_{\sigma[4]} - C_{\sigma[4]})\alpha_{\sigma[4]} = (39 - 36) * 4 + (75 - 70) * 3 = 27$ corresponds to the subsequent breakpoint $(M_2, G_2) = (82, 27)$ on the TWET-makespan trade-off plot, as shown in steps 29–31 of Algorithm 2. Block $B = \{4, 5\}$ is further left shifted by 2 units, encountering Case 2, where $\sigma[4]$ becomes contiguous with $\sigma[3]$, to obtain the schedule $S = \{36, 44, 57, 68, 80\}$ with $TWET(\sigma) = (de_{\sigma[1]} - C_{\sigma[1]})\alpha_{\sigma[1]} + (de_{\sigma[4]} - C_{\sigma[4]})\alpha_{\sigma[4]} + (de_{\sigma[5]} - C_{\sigma[5]})\alpha_{\sigma[5]} = (39 - 36) * 4 + (75 - 68) * 3 + (82 - 80) * 5 = 43$. This corresponds to the subsequent breakpoint $(M_3, G_3) = (80, 43)$ on the trade-off plot. At this point, all the jobs are contiguously scheduled and the block is updated, as shown in steps 33–43 of Algorithm 2. The jobs corresponding to the updated block $B = \{1, 2, 3, 4, 5\}$ are left shifted by the smallest unit of left shift that leads to encountering one of the following cases as shown in steps 17–24 of Algorithm 2.

Case 1: A job with its position identifier in B becomes early (i.e.at $C_{\sigma[2]} = 40$ and $C_{\sigma[3]} = 50$)
Case 2: No idle time exists before the first job $\sigma[1]$ in B (i.e. at $C_{\sigma[1]} = 7$)

In this scenario, Case 1 (i.e. $C_{\sigma[2]} = 40$) is encountered first, and the jobs with position identifiers in B are left shifted by 4 units to obtain the schedule $S = \{32, 40, 53, 64, 76\}$ that corresponds to the subsequent breakpoint $(M_4,\ G_4) = (76, 91)$ on the trade-off plot. Case 1 (i.e. $C_{\sigma[3]} = 50$) is encountered next, and the jobs with position identifiers in B are left shifted by 3 units to obtain the schedule $S = \{29, 37, 50, 61, 73\}$ that corresponds to the subsequent breakpoint $(M_5,\ G_5) = (73, 136)$ on the trade-off plot. Case 2 is encountered next, and the jobs with position identifiers in B are shifted by 22 units to obtain the schedule $S = \{7, 15, 28, 39, 51\}$ that corresponds to the makespan-optimal trade-off point $(M_6,\ G_6) = (51, 598)$, beyond which no left shifting is possible. These breakpoints are saved as shown in steps 29–31 of Algorithm 2. Fig. A.3 shows the piecewise linear convex TWET-makespan trade-off curve generated in the illustration problem for the given σ.

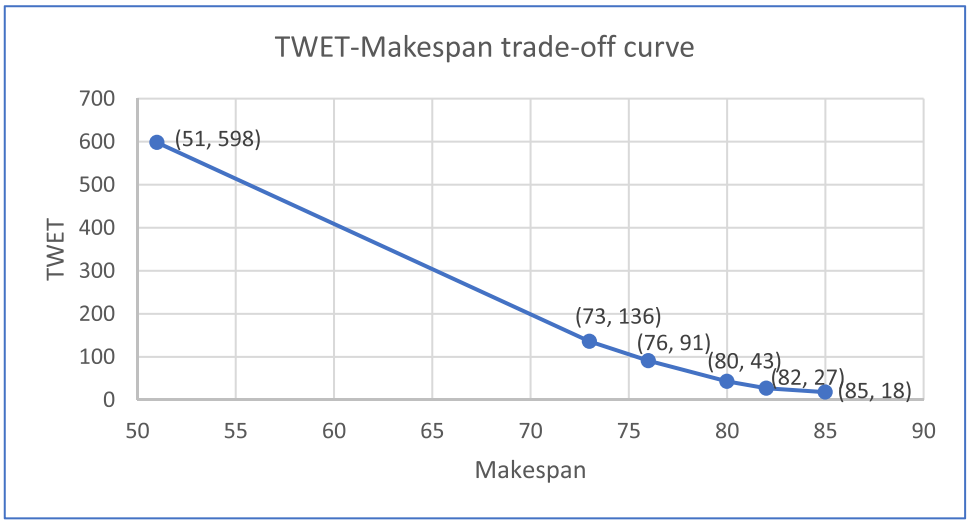


Fig. A.3. TWET-Makespan trade-off curve generated for the illustration problem.

Data availability

Data will be made available on request.

References

[1] Yazdani M, Haghani M. Exploring the evolution of machine scheduling through a computational approach. Eng Appl Artif Intell 2024;133:108572. https://doi.org/10.1016/j.engappai.2024.108572.
[2] Baker KR, Trietsch D. Principles of sequencing and scheduling. 2nd ed. Wiley; 2018.
[3] Morais R, Bulhões T, Subramanian A. Exact and heuristic algorithms for minimizing the makespan on a single machine scheduling problem with sequence-dependent setup times and release dates. Eur J Oper Res 2024;315:442–53. https://doi.org/10.1016/j.ejor.2023.11.024.
[4] Pinedo ML. Scheduling - Theory, algorithms, and systems. 5th ed. Cham: Springer; 2016. https://doi.org/10.1007/978-3-319-26580-3.
[5] Allahverdi A. The third comprehensive survey on scheduling problems with setup times/costs. Eur J Oper Res 2015;246:345–78. https://doi.org/10.1016/j.ejor.2015.04.004.
[6] Baker KR, Scudder GD. Sequencing with earliness and tardiness penalties: a review. Oper Res 1990;38:22–36.
[7] Józefowska J. In: Józefowska J, editor. Just-in-time concept in manufacturing and computer systems. Boston, MA: Springer US; 2007. p. 1–23. https://doi.org/10.1007/978-0-387-71718-0_1. editorJust-In-Time Sched. Model. Algorithms Comput. Manuf. Syst.
[8] Bai B, Wei CM, He HY, Wang JB. Study on single-machine common/slack due-window assignment scheduling with delivery times, variable processing times and outsourcing. Mathematics 2024;12.
[9] Qiu XY, Wang JB. Single-machine scheduling with mixed due-windows and deterioration effects. J Appl Math Comput 2024:1–16.
[10] Wang JB, Lv DY, Wan C. Proportionate flow shop scheduling with job-dependent due windows and position-dependent weights. Asia Pac J Oper Res 2024.
[11] Wang JB, Wang YC, Wan C, Lv DY, Zhang L. Controllable processing time scheduling with total weighted completion time objective and deteriorating jobs. Asia Pac J Oper Res 2024;41:2350026. https://doi.org/10.1142/S0217595923500264.
[12] Lv DY, Wang JB. Single-machine group technology scheduling with resource allocation and slack due window assignment including minmax criterion. J Oper Res Soc 2024:1–17.
[13] Arroyo JEC, dos Santos Ottoni R, dos Santos A. Multi-objective variable neighborhood search algorithms for a just-in-time single machine scheduling problem. In: Proceedings of the 2011 11th international conference on intelligent systems design and applications. IEEE; 2011. p. 1116–21. https://doi.org/10.1109/ISDA.2011.6121808.
[14] Girish BS, Habibullah DJ. Minimizing the total weighted earliness and tardiness for a sequence of operations in job shops. RAIRO Oper Res 2022;56:2621–49. https://doi.org/10.1051/ro/2022124.
[15] Haeussler S, Neuner P, Thürer M. Balancing earliness and tardiness within workload control order release: an assessment by simulation. Flex Serv Manuf J 2023;35:487–508. https://doi.org/10.1007/s10696-021-09440-9.
[16] Pan QK, Wang L, Mao K, Zhao JH, Zhang M. An effective artificial bee colony algorithm for a real-world hybrid flowshop problem in steelmaking process. IEEE Trans Autom Sci Eng 2013;10:307–22. https://doi.org/10.1109/TASE.2012.2204874.
[17] Xu Z, Zheng Z, Gao X. Energy-efficient steelmaking-continuous casting scheduling problem with temperature constraints and its solution using a multi-objective hybrid genetic algorithm with local search. Appl Soft Comput 2020;95:106554. https://doi.org/10.1016/j.asoc.2020.106554.
[18] Rocholl J, Mönch L. Hybrid algorithms for the earliness–tardiness single-machine multiple orders per job scheduling problem with a common due date. RAIRO Oper Res 2018;52:1329–50. https://doi.org/10.1051/ro/2018029. http://www.numdam.org/articles/10.1051/ro/2018029/.
[19] Girish BS. An efficient hybrid particle swarm optimization algorithm in a rolling horizon framework for the aircraft landing problem. Appl Soft Comput 2016;44:200–21. https://doi.org/10.1016/j.asoc.2016.04.011.
[20] Ji M, He Y, Cheng TCE. Single-machine scheduling with periodic maintenance to minimize makespan. Comput Oper Res 2007;34:1764–70. https://doi.org/10.1016/j.cor.2005.05.034.
[21] Lee WC, Wu CC, Liu HC. A note on single-machine makespan problem with general deteriorating function. Int J Adv Manuf Technol 2009;40:1053–6. https://doi.org/10.1007/s00170-008-1421-9.
[22] Low C, Ji M, Hsu CJ, Su CT. Minimizing the makespan in a single machine scheduling problems with flexible and periodic maintenance. Appl Math Model 2010;34:334–42. https://doi.org/10.1016/j.apm.2009.04.014.
[23] Li J, Pan Q, Mao K, Suganthan PN. Solving the steelmaking casting problem using an effective fruit fly optimisation algorithm. Knowl Based Syst 2014;72:28–36. https://doi.org/10.1016/j.knosys.2014.08.022.
[24] Otten M, Braaksma A, Boucherie RJ. Minimizing earliness/tardiness costs on multiple machines with an application to surgery scheduling. Oper Res Heal Care 2019;22:100194. https://doi.org/10.1016/j.orhc.2019.100194.

[25] Yang G, Yu Y, Wang Y, Yin Y, Deng R, Zhang Y. An improved discrete artificial bee colony algorithm for steelmaking and continuous casting scheduling problem. In: Proceedings of the 2023 35th Chinese control and decision conference; 2023. p. 1114–9.
[26] Wan L, Yuan J. Single-machine scheduling to minimize the total earliness and tardiness is strongly NP-hard. Oper Res Lett 2013;41:363–5. https://doi.org/10.1016/j.orl.2013.04.007.
[27] Zhang A, Chen Y, Chen L, Chen G. On the NP-hardness of scheduling with time restrictions. Discret Optim 2018;28:54–62. https://doi.org/10.1016/j.disopt.2017.12.001.
[28] Sterna M. Late and early work scheduling: a survey. Omega 2021;104:102453. https://doi.org/10.1016/j.omega.2021.102453.
[29] Bülbül K, Kaminsky P, Yano C. Flow shop scheduling with earliness, tardiness, and intermediate inventory holding costs. Nav Res Logist 2004;51:407–45. https://doi.org/10.1002/nav.20000.
[30] Kayvanfar V, Mahdavi I, Komaki GM. Single machine scheduling with controllable processing times to minimize total tardiness and earliness. Comput Ind Eng 2013;65:166–75. https://doi.org/10.1016/j.cie.2011.08.019.
[31] Nikabadi SM, Naderi R. A hybrid algorithm for unrelated parallel machines scheduling. Int J Ind Eng Comput 2016;7:681–702. https://doi.org/10.5267/j.ijiec.2016.2.004.
[32] Sadati A, Tavakkoli-Moghaddam R, Naderi B, Mohammadi M. Solving a new multi-objective unrelated parallel machines scheduling problem by hybrid teaching-learning based optimization. Int J Eng Trans B Appl 2017;30:224–33. https://doi.org/10.5829/idosi.ije.2017.30.02b09.
[33] Dhingra A, Chandna P. Multi-objective flow shop scheduling using hybrid simulated annealing. Meas Bus Excell 2010;14:30–41. https://doi.org/10.1108/13683041011074191.
[34] Kayvanfar V, Aalaei A, Hosseininia M, Rajabi M. Unrelated parallel machines scheduling problem with sequence dependent setup times. In: Proceedings of the 2014 international conference on industrial engineering and operations management; 2014. p. 1794–803.
[35] Khanh Van B, Van Hop N. Genetic algorithm with initial sequence for parallel machines scheduling with sequence dependent setup times based on earliness-tardiness. J Ind Prod Eng 2021;38:18–28. https://doi.org/10.1080/21681015.2020.1829111.
[36] Gao J, He G, Wang Y. A new parallel genetic algorithm for solving multiobjective scheduling problems subjected to special process constraint. Int J Adv Manuf Technol 2009;43:151–60. https://doi.org/10.1007/s00170-008-1683-2.
[37] Gao J. A novel artificial immune system for solving multiobjective scheduling problems subject to special process constraint. Comput Ind Eng 2010;58:602–9. https://doi.org/10.1016/j.cie.2009.12.009.
[38] Fakhrzad MB, Sadeghieh A, Emami L. A new multi-objective job shop scheduling with setup times using a hybrid genetic algorithm. Int J Eng Trans B Appl 2013;26:207–18. https://doi.org/10.5829/idosi.ije.2013.26.02b11.
[39] Tajbakhsh Z, Fattahi P, Behnamian J. Multi-objective assembly permutation flow shop scheduling problem: a mathematical model and a meta-heuristic algorithm. J Oper Res Soc 2014;65:1580–92. https://doi.org/10.1057/jors.2013.105.
[40] Abedi M, Seidgar H, Fazlollahtabar H, Bijani R. Bi-objective optimisation for scheduling the identical parallel batch-processing machines with arbitrary job sizes, unequal job release times and capacity limits. Int J Prod Res 2015;53:1680–711. https://doi.org/10.1080/00207543.2014.952795.
[41] Zade AE, Barak S, Maghsoudlou H, Toloo M. Multi-objective optimization for periodic preventive maintenance. In: Proceedings of the 2015 international conference on industrial engineering and systems management. IEEE; 2015. p. 173–82. https://doi.org/10.1109/IESM.2015.7380154.
[42] Zarandi MHF, Kayvanfar V. A bi-objective identical parallel machine scheduling problem with controllable processing times: a just-in-time approach. Int J Adv Manuf Technol 2015;77:545–63. https://doi.org/10.1007/s00170-014-6461-8.
[43] Rad ST, Gholami S, Shafaei R, Seidgar H. Bi-objective optimization for just in time scheduling : application to the two-stage assembly flow shop problem. Qual Eng Prod Optim 2015;1:21–32.
[44] Shahidi-Zadeh B, Evazabadian F, Tavakkoli-Moghaddam R. A new multi-objective scheduling problem on batch parallel machines with maximum allowable incompatibility for jobs. In: Proceedings of the 2015 IEEE international conference on industrial engineering and engineering management. IEEE; 2015. p. 1815–9. https://doi.org/10.1109/IEEM.2015.7385961.
[45] Shahriari M, Shoja N, Zade AE, Barak S, Sharifi M. JIT single machine scheduling problem with periodic preventive maintenance. J Ind Eng Int 2016;12:299–310. https://doi.org/10.1007/s40092-016-0147-9.
[46] Liang X, Ji Y, Huang M. Solving hybrid flow-shop scheduling based on improved multi-objective artificial bee colony algorithm. In: Proceedings of the 2016 2nd international conference on cloud computing and Internet of Things. IEEE; 2016. p. 43–7. https://doi.org/10.1109/CCIOT.2016.7868300.
[47] Shahvari O, Logendran R. A bi-objective batch processing problem with dual-resources on unrelated-parallel machines. Appl Soft Comput 2017;61:174–92. https://doi.org/10.1016/j.asoc.2017.08.014.
[48] Shahidi-Zadeh B, Tavakkoli-Moghaddam R, Taheri-Moghadam A, Rastgar I. Solving a bi-objective unrelated parallel batch processing machines scheduling problem: a comparison study. Comput Oper Res 2017;88:71–90. https://doi.org/10.1016/j.cor.2017.06.019.
[49] Shen J. An uncertain parallel machine problem with deterioration and learning effect. Comput Appl Math 2019;38. https://doi.org/10.1007/s40314-019-0789-5.
[50] Jia Z, Gao L, Zhang X. A new history-guided multi-objective evolutionary algorithm based on decomposition for batching scheduling. Expert Syst Appl 2020;141:112920. https://doi.org/10.1016/j.eswa.2019.112920.
[51] Shao W, Shao Z, Pi D. Multi-objective evolutionary algorithm based on multiple neighborhoods local search for multi-objective distributed hybrid flow shop scheduling problem. Expert Syst Appl 2021;183:115453. https://doi.org/10.1016/j.eswa.2021.115453.
[52] Wei H, Li S, Quan H, Liu D, Rao S, Li C, et al. Unified multi-objective genetic algorithm for energy efficient job shop scheduling. IEEE Access 2021;9:54542–57. https://doi.org/10.1109/ACCESS.2021.3070981.
[53] Ampry E, Komariah A, Kurniady DA, Rafiq M, Priatna A, Ali MH, et al. Multi-objective mathematical modeling for scheduling machines in parallel with batch processors. Ind Eng Manag Syst 2022;21:366–80. https://doi.org/10.7232/iems.2022.21.2.366.
[54] Yue L, Guan Z, Saif U, Zhang F, Wang H. Hybrid Pareto artificial bee colony algorithm for multi-objective single machine group scheduling problem with sequence-dependent setup times and learning effects. Springerplus 2016;5:1593. https://doi.org/10.1186/s40064-016-3265-3.
[55] Duenas A, Petrovic D. Multi-objective genetic algorithm for single machine scheduling problem under fuzziness. Fuzzy Optim Decis Mak 2008;7:87–104. https://doi.org/10.1007/s10700-007-9026-6.
[56] Chen SH, Chen YH. A new two-objective single machine scheduling problem considers a past-sequence-dependent setup time and learning effect. In: Proceedings of the 2018 1st IEEE international conference on knowledge innovation and invention; 2018. p. 309–12. https://doi.org/10.1109/ICKII.2018.8569129.
[57] Jia J, Lu C, Yin L. Energy saving in single-machine scheduling management: an improved multi-objective model based on discrete artificial bee colony algorithm. Symmetry 2022;14. https://doi.org/10.3390/sym14030561 (Basel).
[58] Wang S. Bi-objective optimisation for integrated scheduling of single machine with setup times and preventive maintenance planning. Int J Prod Res 2013;51:3719–33. https://doi.org/10.1080/00207543.2013.765070.
[59] Salmasnia A, Khatami M, Kazemzadeh RB, Zegordi SH. Bi-objective single machine scheduling problem with stochastic processing times. TOP 2015;23:275–97. https://doi.org/10.1007/s11750-014-0337-9.
[60] Molaee E, Moslehi G, Reisi M. Minimizing maximum earliness and number of tardy jobs in the single machine scheduling problem. Comput Math with Appl 2010;60:2909–19. https://doi.org/10.1016/j.camwa.2010.09.046.
[61] Rahmani K, Mahdavi I, Moradi H, Khorshidian H, Solimanpur M. A nondominated ranked genetic algorithm for bi-objective single machine preemptive scheduling in just-in-time environment. Int J Adv Manuf Technol 2011;55:1135–47. https://doi.org/10.1007/s00170-010-3126-0.
[62] Jacquin S, Dufossé F, Jourdan L. An exact algorithm for the bi-objective timing problem. Optim Lett 2018;12:903–14. https://doi.org/10.1007/s11590-018-1237-y.
[63] Babu S, Girish BS. Pareto-optimal front generation for the bi-objective JIT scheduling problems with a piecewise linear trade-off between objectives. Oper Res Perspect 2024;12. https://doi.org/10.1016/j.orp.2024.100299.
[64] Lee CY, Choi JY. A genetic algorithm for job sequencing problems with distinct due dates and general early-tardy penalty weights. Comput Oper Res 1995;22:857–69. https://doi.org/10.1016/0305-0548(94)00073-H.
[65] Wan G, Yen BPC. Tabu search for single machine scheduling with distinct due windows and weighted earliness/tardiness penalties. Eur J Oper Res 2002;142:271–81. https://doi.org/10.1016/S0377-2217(01)00302-2.
[66] Heger J, Voss T. Dynamically adjusting the k-values of the ATCS rule in a flexible flow shop scenario with reinforcement learning. Int J Prod Res 2023;61:147–61. https://doi.org/10.1080/00207543.2021.1943762.
[67] Li Y., Yang Q., Zhou G., Zhao X. Comparison between ATC and ATCS in parallel machine scheduling. ICLEM 2010, 2010, p. 1332–8. 10.1061/41139(387)183.
[68] Pinedo M, Hadavi K. Scheduling: theory, algorithms and systems development. Gaul W, Bachem A, Habenicht W, Runge W, Stahl WW, editors. Operations research proceedings. Berlin, Heidelberg: Springer Berlin Heidelberg; 1992. p. 35–42. 1991.
[69] Liao CJ, Juan HC. An ant colony optimization for single-machine tardiness scheduling with sequence-dependent setups. Comput Oper Res 2007;34:1899–909. https://doi.org/10.1016/j.cor.2005.07.020.
[70] Lee JH, Yu JM, Lee DH. A tabu search algorithm for unrelated parallel machine scheduling with sequence- and machine-dependent setups: minimizing total tardiness. Int J Adv Manuf Technol 2013;69:2081–9. https://doi.org/10.1007/s00170-013-5192-6.
[71] de QTA, Mundim LR. Multiobjective pseudo-variable neighborhood descent for a bicriteria parallel machine scheduling problem with setup time. Int Trans Oper Res 2020;27:1478–500. https://doi.org/10.1111/itor.12738.
[72] Sekkal N, Belkaid F. A multi-objective simulated annealing to solve an identical parallel machine scheduling problem with deterioration effect and resources consumption constraints. J Comb Optim 2020;40:660–96. https://doi.org/10.1007/s10878-020-00607-y.
[73] Xu J, Wu CC, Yin Y, Lin WC. An iterated local search for the multi-objective permutation flowshop scheduling problem with sequence-dependent setup times. Appl Soft Comput 2017;52:39–47. https://doi.org/10.1016/j.asoc.2016.11.031.
[74] Gomes HC, de Assis das Neves F, Souza MJF. Multi-objective metaheuristic algorithms for the resource-constrained project scheduling problem with precedence relations. Comput Oper Res 2014;44:92–104. https://doi.org/10.1016/j.cor.2013.11.002.

[75] Suresh RK, Mohanasundaram KM. Pareto archived simulated annealing for job shop scheduling with multiple objectives. Int J Adv Manuf Technol 2006;29:184–96. https://doi.org/10.1007/s00170-004-2492-x.
[76] Minella G, Ruiz R, Ciavotta M. A review and evaluation of multiobjective algorithms for the flowshop scheduling problem. INFORMS J Comput 2008;20:451–71. https://doi.org/10.1287/ijoc.1070.0258.
[77] Kalyanmoy D. Multi-objective optimization using evolutionary algorithms. John Wiley & Sons; 2001.
[78] Corne DW, Jerram NR, Knowles JD, Oates MJ. PESA-II: region-based selection in evolutionary multiobjective optimization. In: Proceedings of the 3rd annual conference on genetic and evolutionary computation. Morgan Kaufmann Publishers Inc.; 2001. p. 283–90.
[79] Hansen P, Mladenović N, Todosijević R, Hanafi S. Variable neighborhood search: basics and variants. EURO J Comput Optim 2017;5:423–54. https://doi.org/10.1007/s13675-016-0075-x.
[80] Glover F, Laguna M, Martí R. Fundamentals of scatter search and path relinking. Control Cybern 2000;29:653–84.
[81] Resende MGC, Ribeiro CC, Glover F, Martí R. Scatter search and path-relinking: fundamentals, advances, and applications. editors. In: Gendreau M, Potvin JY, editors. Handbook of metaheuristics. Boston, MA: Springer US; 2010. p. 87–107. https://doi.org/10.1007/978-1-4419-1665-5_4.
[82] Resende MGC, Ribeiro CC. GRASP with path-relinking. editors. In: Resende MGC, Ribeiro CC, editors. Optimization by GRASP. New York, NY: Springer New York; 2016. p. 189–204. https://doi.org/10.1007/978-1-4939-6530-4_9. greedy randomized adapt. search proced..
[83] Ho SC, Gendreau M. Path relinking for the vehicle routing problem. J Heuristics 2006;12:55–72. https://doi.org/10.1007/s10732-006-4192-1.
[84] IBM-software. CPLEX callable Library (C API) reference manual. IBM Doc 2021. n.d.
[85] Quinn MJ. Parallel programming in C with MPI and openmp. 1st ed. McGraw Hill Higher Education.; 2003.
[86] Goldberg D. What every computer scientist should know about floating-point arithmetic. ACM Comput Surv 1991;23:5–48. https://doi.org/10.1145/103162.103163.
[87] Audet C, Bigeon J, Cartier D, Le Digabel S, Salomon L. Performance indicators in multiobjective optimization. Eur J Oper Res 2021;292:397–422. https://doi.org/10.1016/j.ejor.2020.11.016.
[88] Riquelme N, Von Lücken C, Baran B. Performance metrics in multi-objective optimization. In: Proceedings of the 2015 latin American computing conference; 2015. p. 1–11. https://doi.org/10.1109/CLEI.2015.7360024.
[89] Zitzler E, Thiele L, Laumanns M, Fonseca CM, da Fonseca VG. Performance assessment of multiobjective optimizers: an analysis and review. IEEE Trans Evol Comput 2003;7:117–32. https://doi.org/10.1109/TEVC.2003.810758.
[90] Mirjalili S, Lewis A. Novel performance metrics for robust multi-objective optimization algorithms. Swarm Evol Comput 2015;21:1–23. https://doi.org/10.1016/j.swevo.2014.10.005.
[91] Bezerra LCT, López-Ibáñez M, Stützle T. An empirical assessment of the properties of inverted generational distance on multi- and many-objective optimization. Trautmann H, Rudolph G, Klamroth K, Schütze O, Wiecek M, Jin Y, et al., editors. Evolutionary multi-criterion optimization. Cham: Springer International Publishing; 2017. p. 31–45.
[92] Schutze O, Esquivel X, Lara A, Coello CAC. Using the averaged Hausdorff distance as a performance measure in evolutionary multiobjective optimization. IEEE Trans Evol Comput 2012;16:504–22. https://doi.org/10.1109/TEVC.2011.2161872.
[93] Van VDA. Multiobjective evolutionary algorithms: classifications, analyses, and new innovations. Air Force Institute of Technology; 1999.
[94] Rizk-Allah RM, Hassanien AE, Slowik A. Multi-objective orthogonal opposition-based crow search algorithm for large-scale multi-objective optimization. Neural Comput Appl 2020;32:13715–46. https://doi.org/10.1007/s00521-020-04779-w.
[95] Coello CAC, Sierra MR. A study of the parallelization of a coevolutionary multi-objective evolutionary algorithm. editors. In: Raúl M, Arroyo-Figueroa G, Sucar LE, Sossa H, editors. MICAI 2004 advances in artificial intelligence. Berlin, Heidelberg: Springer Berlin Heidelberg; 2004. p. 688–97.
[96] Ishibuchi H, Masuda H, Tanigaki Y, Nojima Y. Modified distance calculation in generational distance and inverted generational distance. Lect Notes Comput Sci 2015;9019:110–25. https://doi.org/10.1007/978-3-319-15892-1_8 (Including Subser Lect Notes Artif Intell Lect Notes Bioinformatics).
[97] Jiang S, Ong YS, Zhang J, Feng L. Consistencies and contradictions of performance metrics in multiobjective optimization. IEEE Trans Cybern 2014;44:2391–404. https://doi.org/10.1109/TCYB.2014.2307319.
[98] Derrac J, García S, Molina D, Herrera F. A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms. Swarm Evol Comput 2011;1:3–18. https://doi.org/10.1016/j.swevo.2011.02.002.
[99] Zeng GQ, Chen J, Li LM, Chen MR, Wu L, Dai YX, et al. An improved multi-objective population-based extremal optimization algorithm with polynomial mutation. Inf Sci 2016;330:49–73. https://doi.org/10.1016/j.ins.2015.10.010 (Ny).
[100] Demšar J. Statistical comparisons of classifiers over multiple data sets. J Mach Learn Res 2006;7:1–30.
[101] Martinelli R, Cristina Martins Queiroz Mariano F, Bertini Martins C. Single machine scheduling in make to order environments: a systematic review. Comput Ind Eng 2022;169:108190. https://doi.org/10.1016/j.cie.2022.108190.