Babu, Sona; Girish, B. S.

Article

Pareto-optimal front generation for the bi-objective JIT scheduling problems with a piecewise linear trade-off between objectives

Operations Research Perspectives

Provided in Cooperation with:
Elsevier

Suggested Citation: Babu, Sona; Girish, B. S. (2024) : Pareto-optimal front generation for the bi-objective JIT scheduling problems with a piecewise linear trade-off between objectives, Operations Research Perspectives, ISSN 2214-7160, Elsevier, Amsterdam, Vol. 12, pp. 1-15, https://doi.org/10.1016/j.orp.2024.100299

This Version is available at:
https://hdl.handle.net/10419/325781

Operations Research Perspectives 12 (2024) 100299

Contents lists available at ScienceDirect

Operations Research Perspectives

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

Pareto-optimal front generation for the bi-objective JIT scheduling problems with a piecewise linear trade-off between objectives

Sona Babu, B.S. Girish *

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

ARTICLE INFO

Keywords:
Multi-objective optimization
Single-machine scheduling
Earliness-tardiness objectives
Total flowtime
Pareto front generation
Just-in-time manufacturing

ABSTRACT

This paper proposes a novel method of Pareto front generation from a set of piecewise linear trade-off curves typically encountered in bi-objective just-in-time (JIT) scheduling problems. We have considered the simultaneous minimization of total weighted earliness and tardiness (TWET) and total flowtime (TFT) objectives in a single-machine scheduling problem (SMSP) with distinct job due dates allowing inserted idle times in the schedules. An optimal timing algorithm (OTA) is presented to generate the trade-off curve between TWET and TFT for a given sequence of jobs. The proposed method of Pareto front generation generates a Pareto-optimal front constituted of both line segments and points. Further, we employ a simple local search method to generate sequences of jobs and their respective trade-off curves, which are trimmed and merged to generate the Pareto-optimal front using the proposed method. Computational results obtained using problem instances of different sizes reveal the efficiency of the proposed OTA and the Pareto front generation method over the state-of-the-art methodologies adopted from the literature.

1. Introduction

Sequencing and scheduling problems have attracted considerable attention from researchers in the past five decades, and several scheduling models and solution techniques have been proposed in the literature [1,2]. Scheduling problems generally involve assigning tasks to scarce resources over time to optimize a given set of objectives. The most popular scheduling models in the literature include single-machine scheduling, parallel machine scheduling, flow shop scheduling, job shop scheduling, open shop scheduling and resource-constrained project scheduling [1–3]. The commonly used objectives in these problems include makespan, total weighted flowtime, total weighted tardiness, number of tardy jobs, and total weighted earliness and tardiness [1,2,4].

Much of the research in scheduling in the literature is focused on developing techniques to solve single-objective formulations. The multi-objective formulations, which involve simultaneous optimization of a set of objectives, have gained considerable attention over recent years. A multi-objective function can be formulated as a weighted sum of objectives with known priorities [5]. Multi-objective optimization problems can also be solved using Pareto-based optimization approaches to obtain a set of non-dominated solutions, considering all possible trade-offs between the objectives [6,7]. The state-of-the-art methodologies to solve Pareto-based multi-objective scheduling problems include exact methods like the ε-constraint method and heuristic methods like genetic algorithm, iterated local search, etc. [6,8]. Since most scheduling problems belong to the NP-hard category, heuristic methods are preferred over exact methods to solve the problems in a reasonable computation time [9,10]. The heuristic methods, in general, generate and iteratively improve sequences of jobs (tasks) and their corresponding schedules to obtain a Pareto-optimal front with non-dominated solutions in the Pareto chart representing the objective space [11–14].

In most of the existing works on Pareto-optimal front generation in sequencing and scheduling problems, each sequence of jobs (tasks) generates a single trade-off point with its corresponding objective values on the Pareto chart. This paper considers scenarios in bi-objective scheduling problems where a single sequence of jobs generates a piecewise linear trade-off curve instead of a single trade-off point between the objectives. For instance, Jacquin et al. [15] presented an optimal timing algorithm for the bi-objective optimization of total weighted earliness and total weighted tardiness in a single-machine scheduling problem (SMSP), in which each sequence of jobs results in a piecewise linear convex trade-off curve between the two objectives. This combination of objectives finds its application mostly in

* Corresponding author.
E-mail addresses: sonababu.sct@gmail.com (S. Babu), girish@iist.ac.in (B.S. Girish).

https://doi.org/10.1016/j.orp.2024.100299
Received 30 June 2023; Received in revised form 15 February 2024; Accepted 16 February 2024
Available online 17 February 2024

just-in-time (JIT) manufacturing systems.

The JIT manufacturing systems aim to produce and provide just the ordered quantity of the deliverables at the right place and time [16]. In JIT systems, each job has a processing time and a due date, and whenever a job finishes early or late from its due date, it is penalized in forms such as inventory costs and loss of reputation, respectively. The earliness and tardiness cost functions in JIT scheduling problems are determined based on the job completion times, their respective due dates, and earliness and tardiness penalties. When the earliness and tardiness cost functions are combined as a weighted objective and minimized, the jobs are scheduled closer to their due dates, forming clusters with inserted idle times between the jobs [17,18]. Considering all possible trade-offs between the two conflicting objectives for a given sequence of jobs, as considered by Jacquin et al. [15] for SMSP, it results in a piecewise linear convex trade-off curve due to the inserted idle times. This scenario of Pareto front generation for the simultaneous minimization of two objectives in a scheduling problem resulting in a piecewise linear trade-off curve has not been encountered much in the literature. Nevertheless, the bi-objective optimization problem involving earliness and tardiness in combination is an important scenario in JIT, and it applies to several scheduling problems, including SMSP, job shop scheduling (JSP), flow shop scheduling (FSP), etc.

The total weighted earliness and tardiness (TWET), a vastly explored objective in JIT scheduling, when combined with total flowtime (TFT) or makespan, will also result in piecewise linear trade-off curves. Optimization of TWET and TFT, as single objectives, have been extensively researched in the literature [4,19,20]. However, the combined optimization of these objectives has not been explored adequately. Optimization of TWET results in the jobs being clustered close to their due dates, which leads to the increase in TFT, primarily due to the inserted idle times between jobs. TFT is an important performance measure indicating the work-in-process inventory. Minimizing TFT increases TWET and vice-versa, thus necessitating a trade-off between the two conflicting objectives. Though some of the existing research in the literature considered the above combination of objectives, they optimized the objectives lexicographically, assigning one of the objectives a higher priority over the other. For instance, Arroyo et al. [21] considered the minimization of TWET and TFT in SMSP with distinct job due dates. The optimal solution they obtained is a Pareto front, constituted of trade-off points. Each point on the Pareto front represents an optimum value of total weighted earliness-tardiness (TWET) corresponding to a particular sequence of jobs. The TFT value obtained corresponding to the optimum TWET schedule is adopted as the optimum TFT. This implies that, for a given sequence of jobs, only the TWET objective is optimized and not the TFT. However, if the Pareto-based optimization was performed considering all possible trade-offs between the two objectives, a trade-off curve similar to that in [15] would be obtained instead of a trade-off point. Behnamian and Ghomi [22] considered the simultaneous minimization of the sum of earliness and tardiness and total completion time in a multi-factory parallel machine scheduling problem. They proposed an elastic constraint method and a heuristic algorithm for the generation of a set of Pareto-optimal solutions. However, their approach generates and solves several weighted objective optimization functions leading to discrete points on the Pareto curve. Schulz et al. [23] proposed an iterated local search algorithm to generate a three-dimensional Pareto front to minimize three objectives - makespan, total energy costs, and peak load, where the objectives are prioritized and optimized lexicographically.

The literature has produced several works on finding the Pareto-optimal front of a set of trade-off points [5–7], and the procedure proposed by Kung et al. [24] is one of the most efficient methods. In contrast, not much exploration has been done on finding the Pareto-optimal front of a set of trade-off curves, particularly for the NP-hard scheduling problems. Jacquin et al. [25] presented a genetic algorithm for a bi-objective single-machine scheduling problem to simultaneously minimize the total weighted earliness and total weighted tardiness, where an approximation method of Pareto front generation is proposed to generate the Pareto front from piecewise linear trade-off curves. To the best of our knowledge, no exact method is reported in the literature for generating a Pareto-optimal front for this scenario. This paper presents an exact algorithm for the Pareto-optimal front generation. We have considered the Pareto-based bi-objective optimization in SMSP with the minimization of TWET and TFT as the objectives which is well known to be NP-hard [21,22]. An optimal timing algorithm has been presented to generate the TWET-TFT trade-off curve corresponding to a given sequence of jobs. The proposed exact method for Pareto front generation finds the Pareto-optimal front from a set of trade-off curves constituted of line segments and points.

The literature on bi-objective mixed integer linear programming (BOMILP) has dealt with problems resulting in Pareto-optimal fronts comprising line segments and points [26–28]. The methodologies to generate the Pareto-optimal front in BOMILP are popularly referred to as upper envelop algorithms, which are extensively studied in computational geometry [29–31]. In this paper, we compare the performance of the proposed Pareto front generation method with an upper envelop algorithm adapted from the literature [26–28]. Though the literature on BOMILP uses branch and bound methods to solve problems, such approaches fail to provide results in reasonable computation time in most of the scheduling problems, which are classified as NP-hard. Since the SMSP for the minimization of the TWET objective is proven to be strongly NP-hard [32], the branch and bound-based methods have been used to solve smaller size problem instances, and heuristic approaches have been generally preferred for solving practical size problem instances [22,33,34]. Therefore, we employ a simple local search algorithm, which generates sequences of jobs and iteratively improves them. This requires generating Pareto-optimal fronts several times, thus necessitating an efficient Pareto-optimal front generation method to solve problem instances in reasonable computation time. The literature on BOMILP uses a parametric simplex algorithm to generate trade-off curves comprising line segments. We compare the performance of the optimal timing algorithm proposed in this paper with the parametric simplex algorithm presented in the literature [26–28] in terms of computation time.

The following sections of the paper are organized as follows. Section 2 presents the problem formulation, Section 3 presents the proposed methodologies, Section 4 presents the computational results, and Section 5 concludes with the scope for future work.

2. Problem formulation

The bi-objective single-machine scheduling problem (SMSP) investigated in this paper is described as follows. There are a set of n jobs $J=\{J_1,J_2,\ldots,J_n\}$ to be processed on a single machine that is continuously available. Let i denote the index of the job ($i=1,2,\ldots,n$), and j denote the index of the position of the job in the sequence ($j=1,2,\ldots,n$). Each job i requires a single operation to be performed on the machine. The processing time of job i is given by P_i. All the jobs are available at time zero. The job descriptors are known beforehand and are deterministic. Each job has a due date D_i, the deviations from which would be penalized. If C_i is the scheduled completion time of job i, then the earliness is defined as $E_i = \max(0,\ D_i - C_i)$, and the tardiness is defined as $T_i = \max(0, C_i - D_i)$. If a job gets completed before D_i, an earliness penalty α_i corresponding to E_i would be incurred, whereas if it gets completed after D_i, a tardiness penalty β_i corresponding to T_i would be incurred. The mathematical formulation is as follows.

Decision Variables:

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

C_i = completion time of job i

E_i = earliness of job i

T_i = tardiness of job i

Objective:

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

$$\text{Minimize} \sum_{i=1}^{n}(C_i) \tag{2}$$

Subject to:

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

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

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

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

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

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

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

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

Constraint (3) ensures that exactly one job is assigned to each position in the sequence, and constraint (4) ensures that a job is assigned to exactly one position in the sequence. Constraint (5) ensures that the completion time of the first job in the sequence is greater than or equal to its processing time. Constraint (6) relates the completion time between two successive jobs *i'* and *i*, as per their position in the sequence. The constraint becomes active for all successive pairs of jobs in the sequence and inactive for all non-successive pairs of jobs. *H* represents a large positive integer. Constraint (7) relates the tardiness of each job to its completion time and due date. Constraint (8) relates the earliness of each job with its completion time and due date. The constraints (9) and (10) define the variable bounds.

3. Proposed methodologies

This section first presents the optimal timing algorithm (OTA) for the bi-objective optimization of TWET and TFT. Subsequently, the proposed method of Pareto-optimal front generation for the bi-objective trade-off curves comprising line segments is presented. Lastly, a simple local search algorithm is presented to generate multiple sequences of jobs in an iterative manner to study the performance of the proposed OTA and the Pareto-optimal front generation procedure.

3.1. Optimal timing algorithm for the bi-objective optimization of TWET and TFT

The OTA for the bi-objective optimization problem first finds the optimal TWET schedule for the given sequence of jobs and subsequently generates the optimal trade-off curve between TWET and TFT. A typical TWET-TFT trade-off curve is shown in Fig. 1.

One end of the trade-off curve represents the optimal minimum TWET (T_{opt}) and its corresponding TFT value, and the other end represents the optimal minimum TFT (F_{opt}) and its corresponding TWET value, as shown in Fig. 1. Not only the breakpoints but any point on the line segments constituting the trade-off curve represents the optimal trade-off point between the two objectives for the given sequence of jobs.

The generation of optimal timing schedules for a given sequence of jobs in SMSP for the minimization of TWET objective has been actively researched [35–38], and efficient timing algorithms with $O(n\log n)$ complexity have been presented in the literature. The existing OTA from the literature is adopted to find the optimal TWET schedule in this paper [37]. In an optimal TWET schedule, the jobs are aligned as close as possible to their respective due dates. As a result, the jobs are scheduled exactly at their due dates or form clusters around the due date. This optimal TWET schedule is then iteratively modified to obtain the optimal TWET-TFT trade-off curve. Minimizing TFT from an optimal TWET schedule requires a job or a set of jobs to be left shifted. Left shifting involves reducing the completion time of a job or a set of jobs simultaneously. If no inserted idle times exist in the optimal TWET schedule, then no left shifting of jobs is possible, and the optimal trade-off between the two objectives would be a single point on the Pareto chart. However, if idle time exists, the left shifting of jobs in the optimal TWET schedule generates a trade-off curve between TWET and TFT, as shown in Fig. 1.

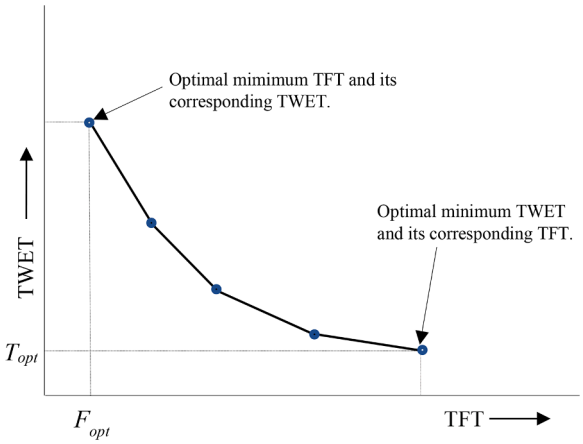

Fig. 1. A typical trade-off curve between TWET and TFT for a given sequence of jobs.

The proposed methodology for generating an optimal TWET-TFT trade-off curve for a given sequence of jobs is as follows. Let σ be a sequence of n number of jobs $\{J_1, J_2, \ldots, J_n\}$ to be scheduled on a single machine for which the optimal TWET schedule obtained by the timing algorithm is given by $S=\{C_1, C_2, \ldots, C_n\}$. σ contains some jobs that are early and some that are tardy. Fig. 2 shows a typical example of an optimal TWET schedule with the jobs clustered around their respective due dates with inserted idle times.

The TWET cost function for σ can be expressed as

$$T(\sigma) = \sum_{i=1}^{n}(\alpha_i \max(0, D_i - C_i) + \beta_i \max(0, C_i - D_i)) \tag{11}$$

If a set of jobs, represented by B ($B \subseteq \sigma$), in the optimal TWET schedule, is selected and left shifted, then the TWET cost function will be $T(\sigma) = T(B) + T(\sigma - B)$, in which $T(\sigma - B)$ will remain constant and $T(B)$ will be given by

$$T(B) = \sum_{i \in B}(\alpha_i \max(0, D_i - C_i) + \beta_i \max(0, C_i - D_i)) \tag{12}$$

To allow for feasible left shifting, the jobs in B must be selected in such a way that if a job i belongs to B, then its preceding contiguously scheduled job i' must also belong to B. This allows all the jobs in B to be simultaneously left shifted by at least one unit of time without violating the constraints (5) and (6) in Section 2. The jobs i' and i, with i' preceding i, are said to be contiguously scheduled if $C_i = C_{i'} + P_i$. Eq. (12) can be rewritten [17] as

$$T(B) = \left(\sum_{i \in TY}\beta_i - \sum_{i \in EY}\alpha_i\right) C_l + \sum_{i \in EY}\alpha_i(D_i + T_i) - \sum_{i \in TY}\beta_i(D_i + T_i) \tag{13}$$

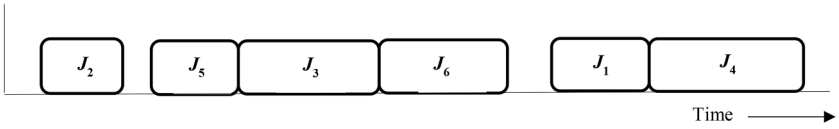

Fig. 2. A typical optimal TWET schedule for a given sequence of jobs.

In Eq. (13), EY and TY denote the set of early and tardy jobs in B, respectively. C_l is the last job in the left shifted set of jobs B. T_i denotes the time gap between the completion time of the last job l and job i in B, i.e., $T_i = C_l - C_i$. If all the jobs in B are left shifted simultaneously by the same amount of time, then $T_i \; \forall i \in B$ will always be constant. The Eq. (13) is in the form of a straight line with a slope given by

$$sl = \sum_{i \in TY} \beta_i - \sum_{i \in EY} \alpha_i \tag{14}$$

Left shifting B beyond the optimal TWET point contributes to a negative slope, as shown in Fig. 3. Left shifting B leads to a decrease in TFT and an increase in TWET from its optimal value. The TWET cost function $T(\sigma)$ increases at the rate of the slope value sl per unit time of left shifting of B. The slope of the cost function $T(\sigma)$ changes whenever a tardy job in B becomes early. As a result, a piecewise linear curve is generated, as shown in Fig. 3. A breakpoint is generated every time the slope of the cost function changes. The block B can be left shifted until a job belonging to B becomes contiguous with a preceding job or an idle time no longer exists preceding the first job when B contains the first job in the sequence σ, whichever occurs first. Left shifting the jobs in B reduces TFT by the rate of N per unit of time, where N denotes the cardinality of the set B.

Since the reduction of TFT due to left shifting causes an increase in TWET, a feasible combination of jobs must be identified for left shifting, which results in the optimal trade-off curve with the smallest slope value. In other words, the block B chosen for left shifting should be the one that causes the smallest increase in TWET with a unit decrease in TFT, determined by the slope of the TWET-TFT trade-off curve, which is given by

$$SL = \frac{\sum_{i \in TY} \beta_i - \sum_{i \in EY} \alpha_i}{N} \tag{15}$$

where N denotes the number of jobs in the block chosen for left shifting. Let B^* represent the optimal block with the smallest absolute value of slope, $|SL|$, among all the blocks formed from the schedule S. The block B^* is left shifted to the nearest breakpoint, which modifies the schedule S. Subsequently, B^* is regenerated with the updated schedule S and further left shifted. B^* is always a single job or a set of contiguously scheduled jobs in the sequence. Repeating this procedure of regenerating B^* with the smallest $|SL|$ at every breakpoint, followed by left shifting until no more idle time exists between jobs, results in the optimal TWET-TFT trade-off curve. The TWET-TFT trade-off curve generated with the above procedure will always be a piecewise linear convex curve, and every point on the curve will be a non-dominated solution for the specific sequence of jobs. This is explained with the following theorems.

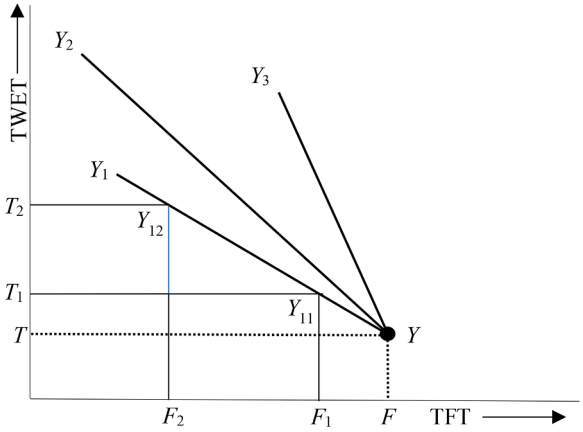

Fig. 4. An illustration of the variations in TWET-TFT slope due to the left shifting of distinct blocks of jobs in σ.

Theorem 1. The optimal block B^* to be left shifted to obtain the optimal TWET-TFT trade-off curve is the block with the smallest absolute ratio of the slope of TWET cost function to the cardinality of the set B^*, given by $|\sum_{i \in TY} \beta_i - \sum_{i \in EY} \alpha_i|/N$.

Proof. Let Y be a breakpoint on the TWET-TFT trade-off curve as shown in Fig. 4 and $S=\{C_1, C_2, \ldots, C_n\}$ be the respective schedule. Let B_1, B_2, and B_3 be the generated blocks in σ, among which the most suitable block B^* to be left shifted must be identified. Let N_1, N_2 and N_3 be the cardinality of B_1, B_2, and B_3, respectively. Let Y_1, Y_2 and Y_3 be the respective breakpoints beyond which the slope of the TWET cost function changes. Obviously, YY_1, with the smallest absolute slope value, is the dominant line segment from point Y to Y_1 on the TWET-TFT trade-off chart compared to line segments YY_2 and YY_3. Therefore, choosing B_1 as B^* would result in an optimal TWET-TFT trade-off curve originating from point Y to Y_1 on the Pareto chart.

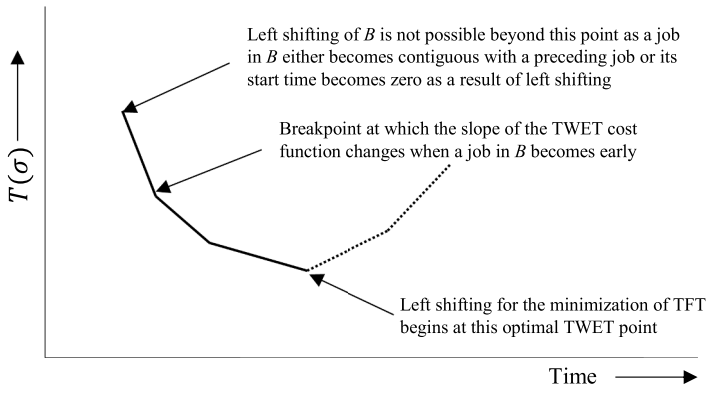

Fig. 3. A typical TWET cost function plot when a set of jobs are left shifted.

The slope of the trade-off curve corresponding to B_1 can be determined as

$$SL = \frac{T_1 - T_2}{F_1 - F_2} \tag{16}$$

where T_1 and T_2 are the TWET values, and F_1 and F_2 are the TFT values corresponding to the points Y_{11} and Y_{12} on the trade-off curve, as shown in Fig. 4. Assuming that the point Y_{12} is obtained by left shifting B_1 by one unit of time from the schedule corresponding to Y_{11}, $F_1 - F_2$ will be equal to N, where N is the cardinality of block B_1. This is because TFT decreases at a rate of N per unit time of left shifting. Left shifting B_1 by one unit of time increases TWET by $\sum_{i\in TY}\beta_i - \sum_{i\in EY}\alpha_i$, which is evident from the Eqs. (13), (14), (15) and their related descriptions. In other words, $T_1 - T_2 = \sum_{i\in TY}\beta_i - \sum_{i\in EY}\alpha_i$. Therefore, the slope of the line segment YY_1 will be given by $SL = \left(\sum_{i\in TY}\beta_i - \sum_{i\in EY}\alpha_i\right)\Big/N$, wherein $\sum_{i\in TY}\beta_i - \sum_{i\in EY}\alpha_i < 0$. Hence, left shifting the block with the smallest absolute ratio of the slope of TWET cost function and the cardinality of the block results in the optimal TWET-TFT trade-off curve between its corresponding breakpoints.

Theorem 2. Suppose there exists more than one disjoint block of contiguously scheduled jobs with the same smallest $|SL|$ available for left shifting in σ. In that case, the optimal block B^* can be created by including all the blocks and left shifted simultaneously, or the individual blocks can be left shifted successively in any order.

Proof. Let A and B be two non-contiguous blocks that can be independently left shifted. Each set, A and B, can either contain a single job or a set of contiguously scheduled jobs. Let sl^A and sl^B be the TWET cost function slope values given by Eq. (14) corresponding to A and B. Let N^A and N^B be the cardinality of the sets A and B, respectively. If either A or B is left shifted, their respective slope of the TWET-TFT trade-off curve according to Eq. (15) will be given by $SL^A = sl^A/N^A$ and $SL^B = sl^B/N^B$, where $sl^A < 0$, $sl^B < 0$, $N^A > 0$ and $N^B > 0$. If both A and B are simultaneously left shifted, the slope of the TWET-TFT trade-off curve will be given by $SL^{A\cup B} = (sl^A + sl^B)/(N^A + N^B)$. Let $|SL^A|$ and $|SL^B|$ denote the absolute value of slopes SL^A and SL^B, respectively. If $|SL^A| \neq |SL^B|$, then $|SL^{A\cup B}| > \min(|SL^A|, |SL^B|)$. If $|SL^A| = |SL^B|$, then based on the property of ratio and proportions [39], $|SL^{A\cup B}| = |SL^A| = |SL^B|$. It can also be generalized that if there are two or more disjoint blocks with the same $|SL|$, then the absolute slope value of the TWET-TFT trade-off curve of all blocks combined will always be equal to that of each of the individual blocks. This implies that if any block with the smallest $|SL|$ comprises a combination of disjoint subsets of a single job or a set of contiguously scheduled jobs, then each subset will have the same $|SL|$ and can be independently left shifted. These disjoint subsets with the same $|SL|$ can be sequentially chosen as the optimal block B^* and left shifted one after the other in any order to obtain the optimal TWET-TFT trade-off curve from a particular breakpoint to its next breakpoint.

Theorem 3. Identifying and left shifting the optimal block B^* with minimum $|SL|$ at every breakpoint on the TWET-TFT trade-off curve always results in a piecewise linear convex curve.

Proof. Each breakpoint on the TWET-TFT trade-off curve results from a change in the slope of the TWET cost function of block B^* due to left shifting. As mentioned previously, the slope of the TWET cost function may change when a tardy job becomes early in the block or when a job in the block becomes contiguous with a preceding job. In the case of a tardy job becoming early in B^*, the slope of the TWET cost function becomes more negative with no change in the cardinality of the block, as evident from Eq. (14). Therefore, the absolute value of the slope of the TWET-TFT curve for the block given by Eq. (15) increases due to left shifting. If a job in B^* becomes contiguous with a preceding job not belonging to the block, the preceding contiguously scheduled jobs must be added to B^* for it to be left shifted. As discussed in the proof of Theorem 2, it can be deduced that $|SL^{A\cup B}| \geq \min(|SL^A|, |SL^B|)$ considering the two cases $|SL^A| = |SL^B|$ and $|SL^A| \neq |SL^B|$, when two blocks A and B are left shifted simultaneously. This implies that the absolute value of the slope $|SL|$ of the TWET-TFT trade-off curve corresponding to B^* increases or remains the same when it is merged with a set of contiguously scheduled jobs preceding it in the sequence. Therefore, the optimal block B^* needs to be identified at each breakpoint for further left shifting, resulting in a piecewise linear convex trade-off curve.

Theorem 4. Every point on the TWET-TFT trade-off curve, generated by identifying and left shifting the optimal block with the minimum $|SL|$ in a sequential manner from the optimal TWET schedule, is a non-dominated solution for the specific sequence of jobs.

Proof. A non-dominated point P on the TWET-TFT Pareto chart corresponds to the best possible minimum value of TWET for a specific value of TFT. To obtain the optimum TFT point (F_{opt}) on the trade-off curve, all the jobs must be left shifted to their earliest possible start time resulting in a schedule with no idle times. If the jobs are selected and left shifted in any order such that the inserted idle times get completely eliminated in the schedule, it always results in the same optimum TFT (F_{opt}) and its corresponding best possible TWET value. However, any intermediate point between the optimum TFT point (F_{opt}) and the optimum TWET point (T_{opt}) can vary from the best possible non-dominated solution based on the order in which the jobs are chosen and left shifted. The increase in TWET and the decrease in TFT contributed by each job in the optimum TFT schedule are the same, irrespective of the order in which the jobs are left shifted. However, to obtain the best possible non-dominated solution for each intermediate point, the jobs should be left shifted such that the increase in TWET should be the minimum for a specific decrease in TFT. Since the procedure to generate the TWET-TFT trade-off curve involves selecting and left shifting the set of jobs in the increasing order of $|SL|$, the increase in TWET for each intermediate point with a specific decrease in TFT will be the minimum. Therefore, the intermediate points generated between the optimum TFT point and the optimum TWET point will always be the best possible non-dominated solutions.

Algorithm 1 and Algorithm 2 show the pseudocode of the optimal timing algorithm for generating the TWET-TFT trade-off curve, which is conceptually similar to the methodology presented in [15]. Algorithm 1 identifies the optimal block B^* for left shifting, and Algorithm 2 left shifts B^* until it becomes contiguous with a preceding job or no idle time exists preceding it for further left shifting. The algorithms use the *SAVE_BREAK_POINT* function to store the TWET and TFT values corresponding to each breakpoint in the order in which it is generated.

Algorithm 1 requires the optimal TWET schedule given by $S=\{C_1, C_2, \ldots, C_n\}$ for the given sequence of jobs σ as input along with other data, including the due date, processing time, and earliness and tardiness penalties corresponding to each job. The TWET and TFT values corresponding to the optimal TWET sequence are determined and stored as the first point using the *SAVE_BREAK_POINT* function, as shown in line 2 of Algorithm 1. σ_i in Algorithm 1 denotes the job in the i^{th} position in the sequence σ. Lines 4-45 find the optimal block B^* by identifying all the feasible blocks containing a single job or a set of contiguously scheduled jobs starting from the first job in σ to the last. If there is no idle time before the first job in σ, lines 7-17 skip all the jobs from forming a block until it finds the first job in σ with an inserted idle time between that job and its preceding job. If no idle time exists between the jobs in σ, then no feasible B^* can be formed for further left shifting. Whenever an idle time is encountered preceding any job i, lines 19-30 create an ordered set B by adding the job i to it, followed by determining the slope of TWET cost function (sl) and the cardinality of B (N). The optimal block B^* is

Algorithm 1
Generation of the TWET-TFT trade-off curve

```
    Input Data: n, σ, P_i, D_i, α_i, β_i, C_i ∀i ∈ σ
1   t ←1,  PSL← 0
2   SAVE_BREAK_POINT(t)
3   do
4       SL ← M                          ►M is a large positive value
5       B ← ϕ, B* ← ϕ                   ► B and B* are ordered sets
6       for (i=1 to n) do
7           if (i=1 and C_σi = P_σi) then
8               for(k=2 to n) do
9                   if (C_σk > C_σk−1 + P_σk) then
10                      Break
11                  else
12                      i ← i + 1
13                      if (i=n) then
14                          go to line 48
15                      end
16                  end
17              end
18          else
19              if (i=1 and C_σi > P_σi) or (i >1 and C_σi > C_σi−1 + P_σi) then
20                  B ← ϕ
21                  if (C_σi > D_σi) then
22                      sl ← β_σi
23                  else
24                      sl ← −α_σi
25                  end
26                  N ← 1,  B ← {i}
27                  if (SL > |sl|/N) then
28                      SL ← |sl|/N
29                      B*← B
30                  end
31              else if (i >1 and C_σi = C_σi−1 + P_σi) then
32                  B ← B ∪ {i}
33                  if (C_σi > D_σi) then
34                      sl ← sl + β_σi
35                  else
36                      sl ← sl−α_σi
37                  end
38                  N ← N + 1
39                  if (SL > |sl|/N) then
40                      SL ← |sl|/N
41                      B*← B
42                  end
43              end
44          end
45      end
46      if (B* ≠ ϕ ) then
47          LEFT_SHIFT(B*, SL)
48      end
49  while (B*≠ ϕ)
```

updated with B if the ratio of the absolute value of sl and N is smaller than the previous ones found. If a job i is contiguous with its preceding job, then block B is updated to include job i besides updating sl and N depending on whether job i is early or tardy, as shown in lines 31-38. Whenever the absolute ratio of sl to N is smaller than the previously found values, the optimal block B^* is updated with B, as shown in lines 39-42. The ordered set B gets reset every time a non-contiguous job is encountered, as shown in lines 19-20. The best slope SL is initially set to a large positive value and is further iteratively updated with the best slope, as shown in lines 28 and 40. The optimal block with the best slope corresponding to the smallest absolute ratio of change in TWET to change in TFT is thus identified, as explained in Theorem 1. The *LEFT_SHIFT* function is then invoked to left shift the optimal block B^* as shown in line 47.

The *LEFT_SHIFT* function shifts all the jobs in B^*, as shown in Algorithm 2. δ denotes the step length by which B^* can be left shifted. δ is initially assigned a large positive integer value. If the first job in B^* is the first job in σ, then δ is updated with the idle time preceding job σ_1 as shown in lines 4-6. On the other hand, if the first job in B^*, σ_i, has a preceding job σ_{i-1}, then δ is updated with the minimum value among the

Algorithm 2
Left shifting procedure to generate the breakpoints in the TWET-TFT trade-off curve

```
1  Function LEFT_SHIFT(B*, SL)
2      t2 ← 0
3      δ ← M                          ▶ M is a large positive integer
4      i ← B*[1]                      ▶ First element of ordered set B*
5      if (i=1) then
6          δ ← C_σ1 − P_σ1
7      else
8          δ ← min(δ, C_σi − C_σi−1 − P_σi)
9      end
10     t2 ← min_{j∈B} (C_σj − D_σj : C_σj > D_σj)
11     δ ← min(δ, t2 : t2 > 0)
12     if (δ > 0) then
13         C_σj ← C_σj − δ ∀ j ∈ B
14         if (SL > PSL) then
15             t←t+1
16             PSL ← SL
17         end
18         SAVE_BREAK_POINT(t)
19     end
20 end
21 Function SAVE_BREAK_POINT(t)
22     G_t ← Σ_{i=1}^{n} (α_i max(0, D_i − C_i) + β_i max(0, C_i − D_i))
23     F_t ← Σ_{i=1}^{n} (C_i)
24 end
```

previous δ and the idle time between the jobs σ_{i-1} and σ_i, as shown in lines 7-9. t_2 in line 10 denotes the step length by which a tardy job in B^* first becomes early on left shifting. δ is updated with the least value among t_2 and the previous value of δ, as shown in line 11. If the δ hence obtained is positive, then all the jobs in B^* are left shifted by δ, as shown in lines 12 and 13. Lines 14-18 determine the TWET and TFT values for the modified schedule S and stores as a new point by incrementing t only if the SL obtained by left shifting B^* is greater than the previous slope value (PSL), resulting in a breakpoint. Otherwise, if SL=PSL, the new point replaces the previous point, thus extending the TWET-TFT line segment with the same slope value on the trade-off curve. Once the left shifting is complete, B^* is regenerated in the subsequent iteration, for which the execution repeats from line 3 of Algorithm 1. The iterations repeat until all the jobs in the sequence become contiguous, and no idle time exists preceding the first job in σ. The breakpoints at which the slope of the TWET-TFT changes during left shifting are stored using the *SAVE_BREAK_POINT* function, as shown in lines 21 to 24 of Algorithm 2.

The loop in lines 6-45 of Algorithm 1 is executed n number of times to find the possibility of generating a block corresponding to each job in the sequence. The computational complexity in left shifting the optimal block B^* in Algorithm 2 is mainly due to lines 10, 13 and 18, which have a time complexity of $O(n)$. The number of times the loop in lines 3-49 of Algorithm 1 is executed depends on the number of times the optimal block is generated and left shifted, which has a time complexity of $O(n)$. Hence, the asymptotic complexity of generating the TWET-TFT trade-off curve can be deduced as $O(n^2)$.

3.2. The proposed Pareto-optimal front generation procedure

In a multi-objective optimization problem involving the simultaneous minimization of two objectives, two points (x_1,y_1) and (x_2,y_2) in the objective space are said to be non-dominated with respect to each other if each point is superior to the other in at least one objective, i.e., either $x_1 < x_2$ and $y_2 > y_1$ or $x_1 > x_2$and $y_2 < y_1$ [10]. By this definition, all the line segments or points in the TWET-TFT trade-off curve will be non-dominated with respect to each other.

The proposed method of Pareto front generation generates a Pareto-optimal front P, from a set of multiple Pareto fronts represented by M_k, where each of the Pareto front $k \in \{1, 2, \ldots, N\}$ represents either a trade-off curve corresponding to a single sequence of jobs or a Pareto front with non-dominated line segments and points generated from multiple sequences of jobs. Let the set of all the Pareto fronts be represented as M such that $M_k \in M$. Let L_k^M be the number of non-dominated line segments on the Pareto front M_k, and let l represent the line segment identifier, such that $l = 1,2,\ldots,L_k^M$. Let (x_{kl1}^M, y_{kl1}^M) and (x_{kl2}^M, y_{kl2}^M) be the two endpoints of a line segment l belonging to M_k. The indexes 1 and 2 in the coordinates indicate the two endpoints of each line segment. Let s_{kl}^M be the slope of the line segment l belonging to M_k. Let L^P be the number of non-dominated line segments added to the Pareto-optimal front P from Pareto fronts in M, and let p represent the line segment identifier, such that $p = 1, 2, \ldots, L^P$. Let (x_{p1}^P, y_{p1}^P) and (x_{p2}^P, y_{p2}^P) be the two endpoints and s_p^P be the slope of the line segment p belonging to P. A point on a trade-off curve or a Pareto front is also represented as a line segment with the same coordinate values for its two endpoints. This is done to generalize the procedure for both line segments and points. The slope values (s_{kl}^M and s_p^P) for the points on a trade-off curve or Pareto front are set as 0. The identifiers of the line segments, l and p, in the respective Pareto fronts, are arranged in the increasing order of their TFT (i.e., their x-coordinate values). For instance, the identifiers of the line segments in M_k are arranged such that $x_{kl1}^M \le x_{kl2}^M, \forall l$ and $x_{kl2}^M \le x_{k,l+1,1}^M$, $l \in \{1,2,\ldots, L_k^M - 1\}$, where $x_{k,l+1,1}^M$ represents the x-coordinate of the first endpoint of the subsequent line segment of l in M_k. Obviously, then $y_{kl1}^M \ge y_{kl2}^M \ \forall l$ and $y_{kl2}^M \ge y_{k,l+1,1}^M$, $l \in \{1,2,\cdots,L_k^M - 1\}$.

Each line segment in M_k has its corresponding sequence of jobs. Let S_k^M be the number of sequences of jobs in the respective Pareto front M_k. In a Pareto front, more than one line segment can be associated with a particular sequence of jobs, and therefore, $S_k^M \le L_k^M$. Let σ_{ks}^M represent the sequences of jobs in M_k, where s is the index representing the sequence of jobs such that $s = 1,2,\ldots, S_k^M$. Let I_{kl}^M be the indexes of the sequence of jobs corresponding to the line segment l, i.e., $I_{kl}^M \in \{1,2,\ldots,S_k^M\}$. Let S^P be

the number of sequences of jobs in P selected from M based on the chosen non-dominated line segments and σ_s^P be the corresponding sequence of jobs, where $s = 1, 2, \ldots, S^P$. Let I_p^P be the index of the sequence of jobs corresponding to line segment p, i.e., $I_p^P \in \{1, 2, \ldots, S^P\}$.

A line segment with endpoints (x_{kl1}^M, y_{kl1}^M) and (x_{kl2}^M, y_{kl2}^M) is non-dominated with respect to another line segment with endpoints $(x_{k'l'1}^M, y_{k'l'1}^M)$ and $(x_{k'l'2}^M, y_{k'l'2}^M)$, if $x_{kl1}^M < x_{k'l'1}^M, x_{kl2}^M \leq x_{k'l'1}^M, y_{kl2}^M \geq y_{k'l'1}^M$ and $y_{kl2}^M > y_{k'l'2}^M$ as shown in Figs. 5(a) and 5(b). Though the conditions $x_{kl2}^M \leq x_{k'l'1}^M, y_{kl2}^M \geq y_{k'l'1}^M$ are sufficient to establish that the line segments are non-dominated with respect to each other, the other two respective conditions $x_{kl1}^M < x_{k'l'1}^M$ and $y_{kl2}^M > y_{k'l'2}^M$ are required if at least one of the entities is a point. For instance, the point l shown in Fig. 5(c) satisfies all four conditions and is non-dominated with respect to the line segment l', whereas the line segment l' dominates point l shown in Fig. 5(d) as it does not satisfy the condition $x_{kl1}^M < x_{k'l'1}^M$.

The procedure to generate the Pareto-optimal front P starts by finding the non-dominated line segment with minimum x-coordinate (TFT) value among all the line segments belonging to Pareto fronts in M. Subsequently, the procedure iteratively finds the non-dominated line segments in the increasing order of the x-coordinate values to generate the entire Pareto-optimal front P. In each iteration, the endpoints of a non-dominated line segment are determined by comparing them with the other line segments in M. In the process, the index of the subsequent non-dominated line segment belonging to M is also found. Algorithm 3 shows the pseudocode to generate the Pareto-optimal front P from M. Lines 3-7 in Algorithm 3 determine the line segment with the minimum x-coordinate value (x^{min}) among all the line segments belonging to M. The endpoint 1 on the first line segment of all the Pareto fronts in M is checked to find x^{min}. If there is more than one line segment with the same x^{min}, then among them, the line segment with the minimum y-coordinate value (y^{min}) is found. If there is more than one line segment with the same x and y-coordinate values ($x^{\min}$, $y^{\min}$), then among them, the line segment with the minimum slope value (s^{min}) is chosen. The selected line segment l belonging to the Pareto front b or a part of it containing the endpoint 1 with coordinates (x^{min}, y^{min}) will be a non-dominated line segment among all the line segments belonging to the Pareto fronts in M.

To find the non-dominated part of the line segment l belonging to the Pareto front b, it is compared with the line segments on each of the Pareto fronts in M, and in the process, finds the subsequent non-dominated line segment t belonging to Pareto front n, as shown in lines 7-53 of Algorithm 3. Let λ_k be the identifier of the line segment in the Pareto front k belonging to M. Initially, the line segments with endpoints ($x^{\min}$, $y^{\min}$) and (x_{bl2}^M, y_{bl2}^M) are assigned as the first line segment to P (i.e. p=1). If there is a subsequent line segment to the line segment l on Pareto front b, then n and t are set as b and l+1, respectively, and ($x^{\min}$, $y^{\min}$) are updated as shown in line 10 of Algorithm 3. Otherwise, n is set as zero, and x^{min} is assigned a large positive value, as shown in line 12 of Algorithm 3. In the subsequent steps, each of the Pareto fronts in M is sequentially accessed, and the corresponding line segments are successively compared with the line segment p, as shown in lines 14 and 15 of Algorithm 3. If the conditions shown in line 16 are satisfied, then the line segment λ_k will be dominated by p, and is therefore skipped. The subsequent line segment on the same Pareto front k is then accessed to check for dominance. If a line segment λ_k satisfies the conditions $x_{k\lambda_k 1}^M \geq x_{p2}^P$ and $y_{k\lambda_k 1}^M \leq y_{p2}^P$ as shown in line 18 of Algorithm 3, then the line segment λ_k is non-dominated with respect to line segment p, and if the conditions in line 19 are satisfied, then (x^{min}, y^{min}) is updated as ($x_{k\lambda_k 1}^M$, $y_{k\lambda_k 1}^M$), and n and t are updated as k and λ_k, respectively. The conditions in line 19 ensure that the updated line segment t belonging to Pareto front n is non-dominated with respect to all the line segments in P and is not dominated by any other line segment in M. If line segment λ_k belonging to the Pareto front k becomes non-dominated with respect to the line segment p, then the line segments from the subsequent Pareto fronts in M are further checked for dominance. Fig. 6 shows different scenarios of dominated and non-dominated line segments established with the conditions in lines 16 and 18 of Algorithm 3.

If the conditions in lines 16 and 18 in Algorithm 3 are not satisfied, then line segment p is checked for intersection with the projection along the y-axis through endpoint 1 of λ_k. If the conditions in line 24 are satisfied, then the intersection of the projection along the y-axis through the coordinates ($x_{k\lambda_k 1}^M$, $y_{k\lambda_k 1}^M$) with the line segment p is found as shown in line 25 in Algorithm 3. If the point of intersection ($x_{k\lambda_k 1}^M$, y) satisfies the conditions in line 26, then (x_{p2}^P, y_{p2}^P) is updated as ($x_{k\lambda_k 1}^M$,y), ($x^{\min}$, $y^{\min}$) is

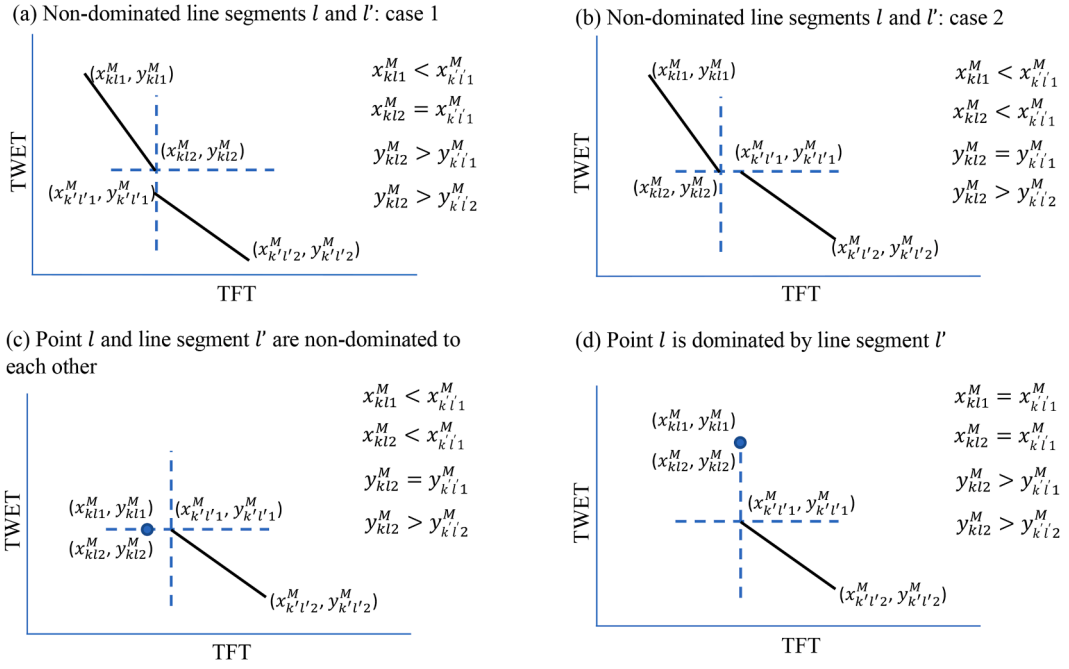

Fig. 5. Typical examples of non-dominated line segments.

Algorithm 3
Proposed method for generation of Pareto-optimal front

Input Data: $N, L_k^M, S_k^M, x_{kl1}^M, y_{kl1}^M, x_{kl2}^M, y_{kl2}^M, s_{kl}^M, I_k^M, \sigma_{ks}^M \quad \forall k = 1,2,\dots,N, \forall l = 1,2,\dots,L_k^M, \forall s = 1,2,\dots,S_k^M$
1 **Initialize:** $x^{min} \leftarrow x_{111}^M, y^{min} \leftarrow y_{111}^M, s^{min} \leftarrow s_{11}^M, p \leftarrow 1, b \leftarrow 1, l \leftarrow 1, \lambda_k \leftarrow 1 \ \forall k = 1,2,\dots,N, S_p^P \leftarrow 0, C_{ks}^M = 0 \ \forall k \forall s$
2 **for** (k=2 *to N*)
3 **if** ($x^{min} > x_{k11}^M$ or ($x^{min} = x_{k11}^M$ and $y^{min} > y_{k11}^M$) or ($x^{min} = x_{k11}^M$ and $y^{min} = y_{k11}^M$ and $s^{min} > s_{k1}^M$))
4 $b \leftarrow k, l \leftarrow 1, \ x^{min} \leftarrow x_{k11}^M, y^{mi\,n} \leftarrow y_{k11}^M, s^{min} \leftarrow s_{k1}^M,$
5 **end**
6 **end**
7 **do**
8 $x_{p1}^P \leftarrow x^{min}, \ y_{p1}^P \leftarrow y^{min}, \ x_{p2}^P \leftarrow x_{bl2}^M, \ y_{p2}^P \leftarrow y_{bl2}^M, \ s_p^P \leftarrow s_{bl}^M, ADD\_SEQUENCES(b, l)$
9 **if** ($l < L_k^M$)
10 $n \leftarrow b, t \leftarrow l + 1, x^{min} \leftarrow x_{b,l+1,1}^M, y^{min} \leftarrow y_{b,l+1,1}^M, s^{min} \leftarrow s_{b,l+1}^M$
11 **else**
12 $n \leftarrow 0, x^{min} \leftarrow H$ // H is a large positive value
13 **end**
14 **for** (k=1 *to* $N: k \neq b$)
15 **while** ($\lambda_k \leq L_k^M$)
16 **if** ($y_{k\lambda_k 2}^M > y_{p1}^P$ or ($x_{k\lambda_k 1}^M \geq x_{p2}^P$ and $y_{k\lambda_k 2}^M \geq y_{p2}^P$))
17 $\lambda_k \leftarrow \lambda_k + 1$
18 **else if** ($x_{k\lambda_k 1}^M \geq x_{p2}^P$ and $y_{k\lambda_k 1}^M \leq y_{p2}^P$)
19 **if** ($x_{k\lambda_k 1}^M < x^{min}$ or ($x_{k\lambda_k 1}^M = x^{min}$ and $y_{k\lambda_k 1}^M < y^{min}$) or ($x_{k\lambda_k 1}^M = x^{min}$ and $y_{k\lambda_k 1}^M = y^{min}$ and $s_{k\lambda_k}^M < s^{min}$))
20 $x^{min} \leftarrow x_{k\lambda_k 1}^M, y^{min} \leftarrow y_{k\lambda_k 1}^M, s^{min} \leftarrow s_{k\lambda_k}^M, n \leftarrow k, \ t \leftarrow \lambda_k$
21 **end**
22 **break**
23 **else**
24 **if** ($x_{p1}^P < x_{k\lambda_k 1}^M$ and $x_{k\lambda_k 1}^M < x_{p2}^P$) //check for intersection on p with projection along y-axis from $(x_{k\lambda_k 1}^M, y_{k\lambda_k 1}^M)$
25 $y \leftarrow s_p^P\left(x_{k\lambda_k 1}^M - x_{p1}^P\right) + y_{p1}^P$
26 **if** $\left(y > y_{k\lambda_k 1}^M\right)$ or ($y = y_{k\lambda_k 1}^M$and $s_{k\lambda_k}^M < s_p^P$ and $s_{k\lambda_k}^M \neq 0$)
27 $x_{p2}^P \leftarrow x_{k\lambda_k 1}^M, \ y_{p2}^P \leftarrow y, x^{min} \leftarrow x_{k\lambda_k 1}^M, y^{min} \leftarrow y_{k\lambda_k 1}^M, s^{min} \leftarrow s_{k\lambda_k}^M, n \leftarrow k, \ t \leftarrow \lambda_k$
28 **break**
29 **end**
30 **end**
31 **if** ($s_p^P > s_{k\lambda_k}^M$ and $s_p^P \neq 0$ and $s_{k\lambda_k}^M \neq 0$) //check for intersection between line segments p and λ_k
32 $x \leftarrow \dfrac{s_p^P x_{p1}^P - s_{k\lambda_k}^M x_{k\lambda_k 1}^M - y_{p1}^P + y_{k\lambda_k 1}^M}{\left(s_p^P - s_{k\lambda_k}^M\right)}$
33 $y \leftarrow s_p^P\left(x - x_{p1}^P\right) + y_{p1}^P$
34 **if** (line segments p and λ_k are intersecting)
35 $x_{p2}^P \leftarrow x, \ y_{p2}^P \leftarrow y, x^{min} \leftarrow x, y^{min} \leftarrow y, s^{min} \leftarrow s_{k\lambda_k}^M, n \leftarrow k, \ t \leftarrow \lambda_k$
36 **break**
37 **end**
38 **end**
39 **if** ($y_{k\lambda_k 1}^M > y_{p2}^P$ and $y_{k\lambda_k 2}^M < y_{p2}^P$) // check for intersection on λ_k with projection along x-axis from (x_{p2}^P, y_{p2}^P)
40 $x \leftarrow \dfrac{\left(y_{p2}^P - y_{k\lambda_k 1}^M\right)}{s_{k\lambda_k}^M} + x_{k\lambda_k 1}^M$
41 **if** ($x > x_{p2}^P$ or ($x = x_{p2}^P$ and $s_p^P \neq 0$))
42 **if** ($x < x^{min}$ or ($x = x^{min}$ and $y_{p2}^P < y^{min}$) or ($x = x^{min}$ and $y_{p2}^P = y^{min}$ and $s_{k\lambda_k}^M < s^{min}$))
43 $x^{min} \leftarrow x, y^{min} \leftarrow y_{p2}^P, s^{min} \leftarrow s_{k\lambda_k}^M, n \leftarrow k, \ t \leftarrow \lambda_k$
44 **end**
45 **break**
46 **end**
47 **end**
48 $\lambda_k \leftarrow \lambda_k + 1$
49 **end**
50 **end**
51 **end**
52 $b \leftarrow n, \ l \leftarrow t, \ p \leftarrow p + 1$
53 **while** ($n > 0$)

updated as ($x_{k\lambda_k 1}^M$, $y_{k\lambda_k 1}^M$), and n and t are updated with identifiers of k and λ_k, respectively. Fig. 7 shows different scenarios of the intersection of the projection of the vertical line through endpoint 1 of λ_k over p, which results in updated endpoint 2 for p. Since the line segment λ_k will be non-dominated with the updated line segment p, the line segments in the other Pareto fronts belonging to M are further checked.

If the conditions in lines 24 and 26 are not satisfied, then the line segment p is checked to see if it intersects with the line segment λ_k between their respective endpoints. The line segments will have a point of intersection only if the slopes of the two line segments are not the same

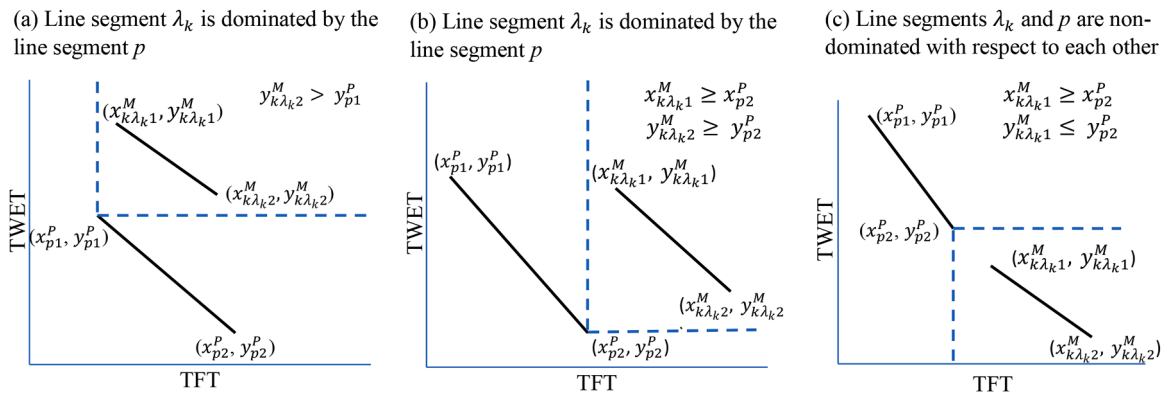

Fig. 6. Typical scenarios of dominated and non-dominated line segments.

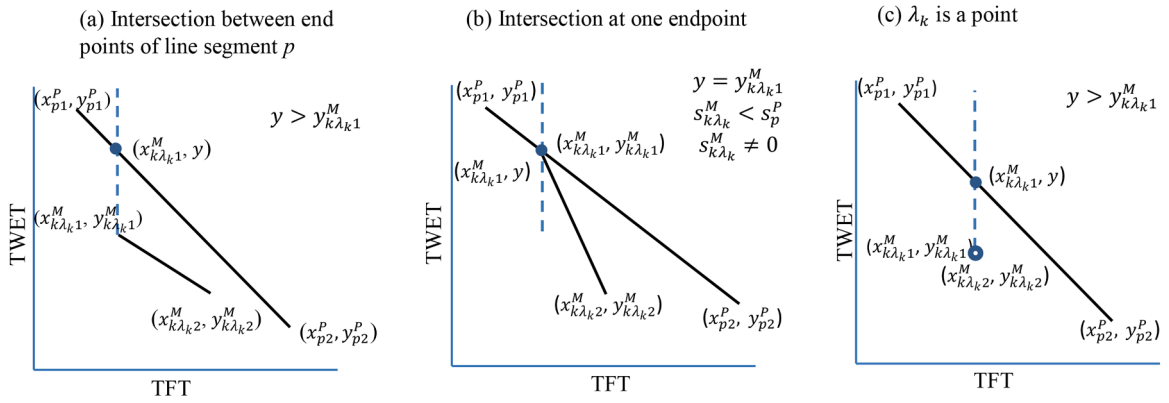

Fig. 7. Typical scenarios of finding the point of intersection of the vertical projection of the first endpoint of line segment λ_k over line segment p.

(i.e., $s_p^P > s_{k\lambda_k}^M$) and they are not points (i.e., $s_p^P \neq 0$ and $s_{k\lambda_k}^M \neq 0$). The case of the intersection of line segments with slope values $s_p^P < s_{k\lambda_k}^M$ will not arise as they would have been already checked for intersection with vertical projection of λ_k over line segment p in lines 24-30 of Algorithm 3. The point of intersection is determined by solving the equations of straight lines corresponding to p and λ_k, as shown in lines 32 and 33 of Algorithm 3. If the two line segments intersect at the point of intersection (x, y) between their respective endpoints, then $(x_{p2}^P,\ y_{p2}^P)$ and $(x^{\min},\ y^{\min})$ are updated with values of (x,y), and n and t are updated with the identifiers of k and λ_k, respectively, as shown in line 35 of Algorithm 3. Since a part of the line segment λ_k will be non-dominated with the updated line segment p, the subsequent line segments on the Pareto front k are not further checked. Fig. 8 shows a typical example of an intersection between the line segments p and λ_k.

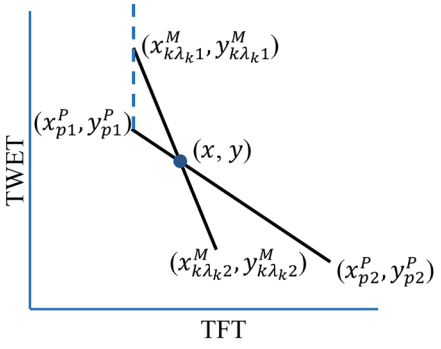

Fig. 8. A typical example of finding the point of intersection between line segments p and λ_k.

If the conditions checked in lines 24, 26, 31 and 34 are not satisfied, then it is checked if the y-coordinate of the two endpoints of the line segment λ_k is on either side of the y-coordinate of the second endpoint of the line segment p. If $y_{k\lambda_k 1}^M > y_{p2}^P$ and $y_{k\lambda_k 2}^M < y_{p2}^P$, then the point (x, y_{p2}^P), which is the point of intersection of the projection of the endpoint $(x_{p2}^P,\ y_{p2}^P)$ parallel to x-axis on the line segment λ_k, is determined using the straight line equation, as shown in line 40 of Algorithm 3. If $x > x_{p2}^P$ or if the conditions $x = x_{p2}^P$ and $s_p^P \neq 0$ are satisfied along with the conditions in line 42, then $(x^{min},\ y^{min})$ is updated as $(x,\ y_{p2}^P)$, and n and t are updated with identifiers of k and λ_k, respectively. Since a part of the line segment λ_k will be non-dominated with respect to the line segment p, the subsequent line segments of the Pareto front k are not checked. Fig. 9 shows different scenarios of the intersection of the projection of the horizontal line segment through endpoint 2 of the line segment p over λ_k.

If none of the above conditions are satisfied, then the line segment λ_k is dominated by p, and it is skipped, as shown in line 48 of Algorithm 3, to further compare with the subsequent line segment on the Pareto front k. The procedure shown in lines 15 to 50 is performed on the line segments of all the Pareto fronts in M as discussed above, and the line segment identifier t corresponding to the Pareto front n is set as l and b, respectively, as shown in line 52 of Algorithm 3. The procedure shown in lines 8 to 52 is repeated with every new non-dominated line segment found until n becomes 0, i.e. no more non-dominated line segments exist to be included in the Pareto front P. The line segments skipped previously were found to be dominated by the line segments already included

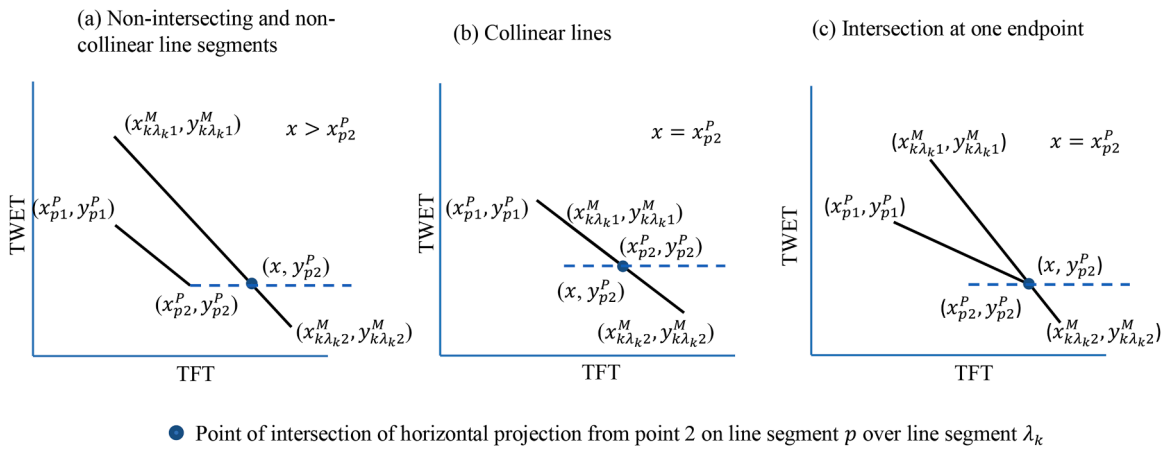

Fig. 9. Typical scenarios of finding the point of intersection of the horizontal projection of the second endpoint of line segment p over λ_k.

in the Pareto-optimal front P.

When line segment l belonging to Pareto front b is compared with the line segments belonging to the Pareto fronts in M to establish its second endpoint and to identify the subsequent non-dominated line segment, the line segments already skipped are not considered again, as shown in the condition of the loop in line 15 of Algorithm 3. This is because the skipped line segments were already dominated by the non-dominated line segments assigned to Pareto-optimal front P in the previous iterations.

As shown in line 8 of Algorithm 3, Algorithm 4 is invoked using the function call *ADD_SEQUENCES* to assign the sequence of jobs corresponding to the identified non-dominated line segment from M to P. Algorithm 4 ensures that a particular sequence of jobs with multiple line segments on its Pareto front is assigned only once from M to P. C^M_{bi} is a binary variable that indicates whether the sequence identifier i corresponding to Pareto front b belonging to M is already assigned to P. O^M_{bi} corresponds to the sequence identifier i on b that stores the sequence identifier of the sequence of jobs in P already assigned to it from M. C^M_{bi} is initialized as 0 and set to 1 when the sequence identifier i on b belonging to M is added to P. A sequence of jobs from M is assigned to P only if C^M_{bi} with respect to that sequence is 0. Otherwise, the sequence identifier of the sequence of jobs belonging to P, which is stored in O^M_{bi}, is directly mapped to the line segment p, as shown in line 7. This procedure ensures that multiple line segments belonging to a particular sequence of jobs in P are all associated with only one sequence identifier in P.

3.3. The local search algorithm

A simple local search (LS) algorithm is employed to generate several sequences of jobs and their corresponding TWET-TFT trade-off curves in order to generate the Pareto-optimal front for each problem instance. The computation time required to generate the TWET-TFT trade-off curves and update the Pareto fronts is used as a performance measure to study how the proposed methods of optimal timing algorithm (OTA) and Pareto front generation perform under different problem settings. The various metaheuristic optimization algorithms presented in the literature for multi-objective optimization, viz. genetic algorithms [10,40], variable neighbourhood search [21], iterated local search [23], etc., iteratively generate multiple sequences of jobs that require efficient Pareto-optimal front generation procedure to find the best possible Pareto front in a short computation time.

Algorithm 5 shows the pseudocode of the local search (LS) method. Initially, a set of *psize* number of solutions, denoted by A, is generated randomly. Each solution represents a sequence of jobs and its corresponding TWET-TFT trade-off curve generated using the procedure presented in Section 3.1. A Pareto-optimal front P is then generated with the TWET-TFT trade-off curves of all the solutions in A using the proposed Pareto-optimal front generation procedure, as shown in line 2 of Algorithm 5. The number of sequences of jobs assigned to the Pareto-optimal front using Algorithm 4 will be less than or equal to *psize*. A set of sequences of jobs not exceeding *psize*, denoted by A in Algorithm 5, is selected randomly from P, and subjected to one iteration of local search, as shown in lines 4-6 of Algorithm 5. In local search, a set of neighbourhoods is generated corresponding to each sequence of jobs $a \in A$ using a pair-wise interchange mechanism. Each neighbourhood corresponding to a sequence of jobs is generated by swapping a pair of jobs such that the difference between their positions in the sequence is within a pre-specified limit, denoted by *swap_limit*. All the possible neighbourhoods are generated corresponding to each sequence of jobs $a \in A$, and the set of all the neigbourhoods generated corresponding to a is denoted by B in Algorithm 5. The TWET-TFT trade-off curve is generated corresponding to each sequence of jobs in B, which is used to update the Pareto-optimal front P. The number of sequences of jobs in the Pareto

Algorithm 4
Assigning the sequences of jobs from M to P without repetition

```
1  Function ADD_SEQUENCES(b, l)
2      i ← I^M_bl
3      if (C^M_bi = 0) then
4          S^P ← S^P + 1
5          C^M_bi ← 1, O^M_bi ← S^P, I^P_p ← S^P, σ^P_{S^P} ← σ^M_bi
6      else
7          I^P_p ← O^M_bi
8      end
9  end
```

Algorithm 5
Pseudocode of the proposed LS algorithm

```
1  A:=InitialPopulationGeneration()
2  P:=GenerateParetoOptimalFront (A)
3  while improvement in solution or maximum number of iterations do
4      A:=SelectPopulation(P)
5      for a ∈ A
6          B:=OneIterationOfLocalSearch(a)
7          P:= UpdateParetoOptimalFront(P, B)
8      end
9  end
```

front P updated with all the solutions in B, as shown in line 7 of Algorithm 5, can exceed *psize*. The process of selecting a set of sequences of jobs to perform one iteration of local search and updating the Pareto-optimal front is repeated until either there is no further improvement in the Pareto-optimal front or a maximum number of iterations, denoted by *max_iter*, is reached.

4. Computational study

This section presents a computational study of the proposed methodologies presented in Section 3. The performance of the proposed methods is evaluated using a set of 24 test instances with the number of jobs varying from 20 to 100. The algorithms were coded in C language and executed using Visual C++ on a PC running Windows 11 with a 3.6 GHz Intel Core i7-9700K octa-core processor and 16GB RAM. The parameters of the LS algorithm, *psize, swap_limit* and *max_iter* are set as 100, 30 and 100, respectively, for the computational study. The arithmetic calculations with the floating-point variables, which include the coordinates of the endpoints of the line segments generated in the proposed Pareto-front generation method, are susceptible to errors [41]. Therefore, the comparison operators implemented with the floating point variables in the C code were restricted to four decimal places. The */fp:strict* option was enabled in the Visual C++ compiler to minimize the floating-point arithmetic errors [42].

This section first presents the procedure used for the generation of test instances. Subsequently, two state-of-the-art methodologies from the literature, the parametric simplex algorithm and the upper envelop algorithm, are discussed, which are adopted to compare with the performance of the proposed timing algorithm and the Pareto-optimal front generation procedure, respectively. Finally, a computational study compares the performance of the proposed methodologies with the methodologies adopted from the literature based on their computation times. Since the proposed optimal timing algorithm and the parametric simplex method adopted from the literature are exact methods, they generate the same TWET-TFT trade-off curve for a given sequence of jobs in a problem instance. Similarly, the proposed Pareto-optimal front generation procedure and the adopted upper envelop algorithm from the literature generate the same Pareto-optimal front from a given set of Pareto fronts, as they are exact methods.

4.1. Test instances

The test instances were generated based on the procedure adopted from [21,25,43]. The number of jobs was chosen from the set n={20,30, 40,50,75,100}. The processing time, tardiness penalty, and earliness penalty for each job are generated uniformly in the interval [1,100], [20,50], and [1,30], respectively. The due date D_i is uniformly generated in the range $[(1-T-RDD/2)TP,(1-T+RDD/2)TP]$, where TP denotes the sum of processing time of all the jobs in a particular problem instance, T denotes the tardiness parameter, and RDD denotes the relative range of the due dates. T and RDD take values from the sets {0.1, 0.3} and {0.8, 1.2} respectively. For each n, a set of 4 test instances are generated with each pair of (T, RDD), thus generating a total of 4×6=24 problem instances. The test instances are named in the pattern P*n*_T*p*_R*q*, where p and q denote the T and RDD values, respectively.

4.2. Methodologies adopted from the literature for performance comparison

To study the performance of the proposed optimal timing algorithm (OTA) presented in Section 3.1, we adopted the parametric simplex algorithm (PSA) presented in [26,44] to generate the TWET-TFT trade-off curve for a given sequence of jobs. The adopted PSA initially optimizes the linear programming formulation of the problem with a fixed sequence of jobs for the TWET objective, and the corresponding optimal TFT is determined. This results in the first point on the TWET-TFT trade-off curve. The basis of the optimal simplex table is then iteratively changed by choosing an entering variable that results in optimal breakpoints on the TWET-TFT trade-off curve. The algorithm was implemented using IBM ILOG CPLEX ver 12.7.1 callable libraries [45], which provides appropriate routines for iteratively modifying the basis and setting iteration limits to obtain the required breakpoints on the TWET-TFT trade-off curve. We used the default settings for the parameters in the CPLEX solver to generate the breakpoints on the trade-off curves. The job sequences corresponding to the non-dominated line segments on the final Pareto-optimal front obtained using the local search algorithm were used to generate the trade-off curves using the proposed OTA and the PSA adopted from the literature, and their performance is compared in terms of the computation time.

To study the performance of the proposed Pareto-optimal front generation procedure presented in Section 3.2, we adapted the upper envelop algorithm (UEA) proposed in [26] to generate the Pareto-optimal front P from the set of multiple Pareto fronts in M. The UEA was originally proposed for a bi-objective optimization problem involving the maximization of two objectives. Since the problem discussed in this paper considers the minimization of two objectives, TWET and TFT, the UEA has been suitably modified. The UEA follows a divide-and-conquer approach to generate a Pareto-optimal front. It initially generates vertical and horizontal line segments through endpoint 1 and endpoint 2, respectively, of all the line segments belonging to the Pareto fronts in M. The vertical and horizontal line segments extend from the endpoints of line segments to the $\max_{k\in\{1,2,..N\}}\{y^M_{k11}\}$ and $\max_{k\in\{1,2,..N)}\{x^M_{kL^M_k 2}\}$ points, respectively. The vertical and horizontal line segments are not required to be generated through the breakpoints between the line segments on a Pareto front, i.e. if endpoint 2 of a line segment is the same as the endpoint 1 of the subsequent line segment on the same Pareto front [28]. Subsequently, the intersection points between the line segments of all the Pareto fronts in M are found, including the horizontal and vertical line segments, and the line segments are split at their point of intersection. The coordinates of the

endpoints, including the breakpoints and the intersection points found between the line segments, are sorted in the increasing order of the x-coordinate values. The sub-intervals formed between the sorted coordinates are then explored to find the non-dominated line segments constituting the Pareto-optimal front P. The horizontal and vertical line segments are discarded when the non-dominated line segments are identified. The readers may refer to the detailed descriptions and illustrations presented in [26–28] for a better understanding of the algorithm.

It is evident from the above description that the adapted UEA generates a large number of intersection points. On the other hand, the proposed Pareto front generation procedure finds the intersection points only between the identified non-dominated line segment l on Pareto-front b and the line segments belonging to the Pareto fronts in M (refer to the notations and the procedure in Section 3.2). The UEA also requires generating a sorted list of line segments after they are split at the intersection points. The number of line segments significantly increases after they are split, which increases the computational effort when line segments are identified to be included in the Pareto-optimal front P. As described in [26], the procedure of finding the non-dominated line segments in each of the sub-intervals was parallelized using OpenMP [46] to reduce the computation time.

4.3. Performance comparison

Table 1 shows the comparison between the proposed Pareto front generation (PFG) method and the UEA in terms of the computation time required to generate the Pareto-optimal fronts within the local search (LS) algorithm for each of the test instances. The PFG and UEA were successively invoked each time a Pareto front was generated within the LS algorithm. Therefore, exactly the same sequences of jobs and the corresponding trade-off curves generated within the local search procedure were used for the comparison of the Pareto front generation methods. The number of times the Pareto-optimal fronts are generated or updated for each test instance is represented as NUM_POF in Table 1. TOT_POF indicates the total computation time required to generate all the NUM_POF number of Pareto fronts from the beginning of the run of the respective algorithms. TOT_POF includes only the computation time for the generation of Pareto-optimal fronts and does not include the computation time required for generating sequences of jobs or the trade-off curves between objectives using the timing algorithm. The number of line segments (L^P) on the final Pareto-optimal front generated for each test instance with the two Pareto-front generation methods and the corresponding number of sequences of jobs (S^P) are also shown in Table 1.

Table 1 also compares the proposed OTA and the PSA in terms of the computation time required to generate the TWET-TFT trade-off curves with the sequences of jobs on the final Pareto-optimal front obtained with the Pareto front generation methods. TOT_TOC indicates the total time required to generate all the S^P number of trade-off curves with the two methods, OTA and PSA.

We use times faster (TF) as a performance measure to compare the two Pareto front generation methods [17]. TF indicates how many times a particular method is faster than the other method used for comparison. TF_POF, shown in Table 1, is the ratio of TOT_POF required by UEA to the TOT_POF required by PFG. TF_POF specifies how many times the PFG method is faster than UEA, which indicates the improvement in performance if PFG is used instead of UEA for Pareto front generation. We also use TF as a performance measure to compare the TWET-TFT trade-off curve generation methods, OTA and PSA. TF_TOC, as shown in Table 1, is determined as the ratio of TOT_TOC required by PSA to the TOT_TOC required by OTA, which indicates the improvement in performance if OTA is used instead of PSA for generating the TWET-TFT trade-off curves.

The TF values for the Pareto-optimal front generation procedures (TF_POF) reveal that the proposed PFG method performs approx. 3 to 30 times faster than the UEA, with the average TF value for all the instances being approx. 10. The TF values corresponding to the TWET-TFT trade-off generation procedures (TF_TOC) reveal that the proposed OTA performs approx. 30 to 200 times faster than the PSA, with the average TF value for all the instances being approx. 90. The comparison of TOT values between PFG and UEA reveals that the UEA requires much higher computation time than PFG when used in applications that require generating Pareto-optimal fronts several times, as in metaheuristic algorithms. These results clearly reveal that the proposed PFG and OTA outperform the UEA and PSA algorithms, respectively, and are best

Table 1
Computational results obtained with the methods of Pareto front generation and the methods of generating TWET-TFT trade-off curves

Sl No.	No. of jobs	Problem instances	S^P	L^P	NUM_POF	TOT_POF (in seconds)		TF_POF	TOT_TOC (in seconds)		TF_TOC
						PFG	UEA		OTA	PSA	
1	20	P20_T0.1_R0.8	502	790	3006	5.763	58.039	10.071	0.005795	0.840750	145.082
2	20	P20_ T0.1_R1.2	964	1213	6332	19.430	462.871	23.822	0.006729	1.172972	174.316
3	20	P20_ T0.3_R0.8	1761	1782	6183	26.015	751.679	28.894	0.011958	1.630334	136.338
4	20	P20_ T0.3_R1.2	933	999	4288	7.276	206.389	28.366	0.003389	0.736851	217.424
5	30	P30_T0.1_R0.8	1663	2007	9267	151.678	1213.724	8.002	0.032294	3.604642	111.620
6	30	P30_ T0.1_R1.2	1183	1768	9046	126.438	731.749	5.787	0.026984	3.085761	114.355
7	30	P30_ T0.3_R0.8	2351	2479	9934	148.611	2164.920	14.568	0.028625	3.386520	118.306
8	30	P30_ T0.3_R1.2	944	1231	7863	64.347	393.639	6.117	0.014017	1.833010	130.770
9	40	P40_T0.1_R0.8	1172	1196	9748	235.567	815.321	3.461	0.051459	4.401800	85.540
10	40	P40_ T0.1_R1.2	1883	1920	9856	277.625	1280.351	4.612	0.069346	5.512491	79.493
11	40	P40_ T0.3_R0.8	2009	2026	9870	251.605	1431.464	5.689	0.052458	4.964838	94.644
12	40	P40_ T0.3_R1.2	2033	2052	9889	196.986	1181.586	5.998	0.047180	5.183847	109.874
13	50	P50_T0.1_R0.8	1080	1105	9849	292.111	1004.307	3.438	0.067158	4.845291	72.148
14	50	P50_ T0.1_R1.2	1311	1330	9824	325.291	1115.683	3.430	0.077290	6.164948	79.764
15	50	P50_ T0.3_R0.8	1656	1658	9903	126.251	1900.578	15.054	0.033320	3.390362	101.752
16	50	P50_ T0.3_R1.2	1846	1856	9881	186.206	941.897	5.058	0.049678	5.421255	109.128
17	75	P75_T0.1_R0.8	672	705	9878	473.569	3341.435	7.056	0.144210	5.640768	39.115
18	75	P75_ T0.1_R1.2	503	547	9784	487.975	3658.128	7.497	0.141283	4.574282	32.377
19	75	P75_ T0.3_R0.8	671	707	9944	466.410	2476.92	5.311	0.167308	6.850280	40.944
20	75	P75_ T0.3_R1.2	570	598	9800	426.958	1770.7135	4.147	0.109786	4.749684	43.263
21	100	P100_T0.1_R0.8	386	438	9895	505.062	6540.153	12.949	0.142584	5.928711	41.580
22	100	P100_ T0.1_R1.2	323	346	9884	419.247	4859.435	11.591	0.088240	4.199653	47.594
23	100	P100_ T0.3_R0.8	403	432	9918	521.168	4915.486	9.432	0.108374	5.209444	48.069
24	100	P100_ T0.3_R1.2	396	412	9949	336.649	3025.861	8.988	0.065366	4.065011	62.188
Average						253.260	1926.764	9.972	0.064368	4.058063	93.154

suited for implementation within metaheuristic algorithms.

5. Conclusions

In this paper, we proposed a Pareto front generation method to generate the Pareto-optimal front from a set of piecewise linear trade-off curves, which is typically encountered in bi-objective just-in-time (JIT) scheduling problems. We considered the simultaneous minimization of the total weighted earliness and tardiness (TWET) and total flowtime (TFT) objectives in a single-machine scheduling problem (SMSP) with distinct job due dates and allowing inserted idle times in the schedules. We proposed an optimal timing algorithm (OTA) to generate a piecewise linear trade-off curve between the TWET-TFT objectives for a given sequence of jobs. The proposed method of Pareto front generation (PFG) is an exact method that generates a Pareto-optimal front from a set of Pareto fronts constituted of line segments and points. The performance of the proposed PFG was compared with an upper envelop algorithm (UEA), and the performance of the proposed OTA was compared with a parametric simplex algorithm (PSA), both adopted from the literature. The computational study performed on 24 problem instances with the number of jobs varying from 20 to 100 jobs reveals that the proposed PFA and OTA outperform the respective methodologies, UEA and PSA, adopted from the literature. Therefore, the proposed methods are best suited for implementation within heuristic and metaheuristic algorithms, which require the generation of TWET-TFT trade-off curves and Pareto-optimal fronts a large number of times.

To the best of our knowledge, this is the first reported exact method for the generation of a Pareto-optimal front from a set of piecewise linear trade-off curves in NP-hard scheduling problems. The proposed Pareto front generation method can be suitably modified and extended to the bi-objective mixed integer linear programming problems. A future research direction would be to develop efficient heuristic and metaheuristic algorithms to solve the Pareto-based bi-objective optimization of TFT and TWET using the Pareto front generation method presented in this paper and perform computational studies. Future research can also be directed towards developing timing algorithms for the bi-objective optimization of other combinations of objectives, viz. TWET-makespan, total weighted earliness-makespan, etc., and performing computational studies with the Pareto front generation method proposed in this paper. Future research can also be directed towards extending the proposed method of Pareto front generation to other scheduling problems in a JIT environment, including flow shop, job shop, open shop, and project scheduling problem, and performing computational studies.

CRediT authorship contribution statement

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

Data availability

Data will be made available on request.

Acknowledgements

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

References

[1] Baker KR, Trietsch D. Principles of sequencing and scheduling. 2nd ed. John Wiley & Sons, Inc.; 2018. https://doi.org/10.1002/9781119262602.
[2] Pinedo ML. Scheduling - Theory, Algorithms, and Systems. 5th ed. Cham: Springer; 2016. https://doi.org/10.1007/978-3-319-26580-3.
[3] Lawler EL, Lenstra JK, Rinnooy Kan AHG, Shmoys DB. Sequencing and scheduling: Algorithms and complexity. In: Graves SC, Rinnooy Kan AHG, Zipkin PH, editors. Logist. Prod. Invent., vol. 4. Elsevier; 1993. p. 445–522. https://doi.org/10.1016/S0927-0507(05)80189-6.
[4] Sterna M. Late and early work scheduling: A survey. Omega 2021;104:102453. https://doi.org/10.1016/j.omega.2021.102453.
[5] Gunantara N. A review of multi-objective optimization: Methods and its applications. Cogent Eng 2018;5:1502242. https://doi.org/10.1080/23311916.2018.1502242.
[6] Cho J-H, Wang Y, Chen I-R, Chan KS, Swami A. A Survey on Modeling and Optimizing Multi-Objective Systems. IEEE Commun Surv Tutorials 2017;19:1867–901. https://doi.org/10.1109/COMST.2017.2698366.
[7] Taha K. Methods That Optimize Multi-Objective Problems: A Survey and Experimental Evaluation. IEEE Access 2020;8:80855–78. https://doi.org/10.1109/ACCESS.2020.2989219.
[8] Halffmann P, Schäfer LE, Dächert K, Klamroth K, Ruzika S. Exact algorithms for multiobjective linear optimization problems with integer variables: A state of the art survey. J Multi-Criteria Decis Anal 2022;29:341–63. https://doi.org/10.1002/mcda.1780.
[9] Coello Coello CA, Brambila SG, Gamboa JF, Tapia MGC, Gómez RH. Evolutionary multiobjective optimization: open research areas and some challenges lying ahead. Complex Intell Syst 2020;6:221–36. https://doi.org/10.1007/s40747-019-0113-4.
[10] Deb K. Multi-objective optimization Using evolutionary algorithms. John Wiley & Sons, Ltd; 2001.
[11] Ojstersek R, Brezocnik M, Buchmeister B. Multi-objective optimization of production scheduling with evolutionary computation: A review. Int J Ind Eng Comput 2020;11:359–76. https://doi.org/10.5267/j.ijiec.2020.1.003.
[12] Dimopoulos C. A review of evolutionary multiobjective optimization applications in the area of production research. In: Proc. 2004 Congr. Evol. Comput. (IEEE Cat. No.04TH8753), vol. 2; 2004. p. 1487–94. https://doi.org/10.1109/CEC.2004.1331072.
[13] Wang X-J, Zhang C-Y, Gao L, Li P-G. A Survey and Future Trend of Study on Multi-Objective Scheduling. In: 2008 Fourth Int. Conf. Nat. Comput., vol. 6; 2008. p. 382–91. https://doi.org/10.1109/ICNC.2008.817.
[14] Gen M, Zhang W, Lin L. Survey of Evolutionary Algorithms in Advanced Planning and Scheduling. J Korean Inst Ind Eng 2009;35:15–39.
[15] 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.
[16] Józefowska J. Just-in-Time scheduling-models and algorithms for computer and manufacturing systems. 1st ed. New York, NY: Springer; 2007. https://doi.org/10.1007/978-0-387-71718-0.
[17] Girish BS, Habibullah Dileeplal J. 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.
[18] Chen J-Y, Lin S-F. Minimizing weighted earliness and tardiness penalties in single-machine scheduling with idle time permitted. Nav Res Logist 2002;49:760–80. https://doi.org/10.1002/nav.10039.
[19] Lu C-C, Lin S-W, Ying K-C. Robust scheduling on a single machine to minimize total flow time. Comput Oper Res 2012;39:1682–91. https://doi.org/10.1016/j.cor.2011.10.003.
[20] Chen WJ. Minimizing total flow time in the single-machine scheduling problem with periodic maintenance. J Oper Res Soc 2006;57:410–5. https://doi.org/10.1057/palgrave.jors.2601998.
[21] Arroyo JEC, Ottoni R dos S, Oliveira A de P. Multi-objective Variable Neighborhood Search Algorithms for a Single Machine Scheduling Problem with Distinct due Windows. Electron Notes Theor Comput Sci 2011;281:5–19. https://doi.org/10.1016/j.entcs.2011.11.022.
[22] Behnamian J, Ghomi SMTF. Multi-objective multi-factory scheduling. RAIRO-Operations Res 2021;55:S1447–67. https://doi.org/10.1051/ro/2020044.
[23] Schulz S, Neufeld JS, Buscher U. A multi-objective iterated local search algorithm for comprehensive energy-aware hybrid flow shop scheduling. J Clean Prod 2019;224:421–34. https://doi.org/10.1016/j.jclepro.2019.03.155.
[24] Kung HT, Luccio F, Preparata FP. On Finding the Maxima of a Set of Vectors. J ACM 1975;22:469–76. https://doi.org/10.1145/321906.321910.
[25] Jacquin S, Allart E, Dufossé F, Jourdan L. Decoder-based evolutionary algorithm for bi-objective just-in-time single-machine job-shop. 2016 IEEE Symp. Ser. Comput. Intell. 2016:1–8. https://doi.org/10.1109/SSCI.2016.7850054.
[26] Soylu B. Heuristic approaches for biobjective mixed 0–1 integer linear programming problems. Eur J Oper Res 2015;245:690–703. https://doi.org/10.1016/j.ejor.2015.04.010.
[27] Soylu B, Yıldız GB. An exact algorithm for biobjective mixed integer linear programming problems. Comput Oper Res 2016;72:204–13. https://doi.org/10.1016/j.cor.2016.03.001.
[28] Soylu B. The search-and-remove algorithm for biobjective mixed-integer linear programming problems. Eur J Oper Res 2018;268:281–99. https://doi.org/10.1016/j.ejor.2018.01.026.

[29] Edelsbrunner H, Guibas LJ, Sharir M. The upper envelope of piecewise linear functions: Algorithms and applications. Discrete Comput Geom 1989;4:311–36. https://doi.org/10.1007/BF02187733.
[30] Hershberger J. Finding the upper envelope of n line segments in O(n log n) time. Inf Process Lett 1989;33:169–74. https://doi.org/10.1016/0020-0190(89)90136-1.
[31] Chen W, Wada K. On computing the upper envelope of segments in parallel. In: Proceedings. 1998 Int. Conf. Parallel Process. (Cat. No.98EX205; 1998. p. 253–60. https://doi.org/10.1109/ICPP.1998.708493.
[32] 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.
[33] Ronconi D, Kawamura M. The single machine earliness and tardiness scheduling problem: Lower bounds and a branch-and-bound algorithm. Comput Appl Math 2010;29:107–24. https://doi.org/10.1590/S1807-03022010000200002.
[34] Sourd F, Kedad-Sidhoum S. A faster branch-and-bound algorithm for the earliness-tardiness scheduling problem. J Sched 2008;11:49–58. https://doi.org/10.1007/s10951-007-0048-2.
[35] Szwarc W, Mukhopadhyay SK. Optimal timing schedules in earliness-tardiness single machine sequencing. Nav Res Logist 1995;42:1109–14. https://doi.org/10.1002/1520-6750(199510)42:7<1109::AID-NAV3220420709>3.0.CO;2-5.
[36] Wan G, BP-C Yen. Tabu search for single machine scheduling with distinct due windows and weighted earliness/tardiness penalties. Eur J Oper Res 2002;142:271–81.
[37] 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.
[38] Feng G, Lau HC. Efficient algorithms for machine scheduling problems with earliness and tardiness penalties. Ann Oper Res 2008;159:83–95. https://doi.org/10.1007/s10479-007-0284-z.
[39] Ben-Chaim D, Keret Y, Ilany B-S. Ratio and Proportion. SensePublishers Rotterdam; 2012. https://doi.org/10.1007/978-94-6091-784-4.
[40] Bagchi TP. Multiobjective Scheduling by Genetic Algorithms. 1st ed. New York, NY: Springer; 2012. https://doi.org/10.1007/978-1-4615-5237-6.
[41] 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.
[42] Microsoft. Microsoft C++, C, and Assembler documentation 2022. https://learn.microsoft.com/en-us/cpp/build/reference/fp-specify-floating-point-behavior?view=msvc-170#strict (accessed June 28, 2023).
[43] 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.
[44] Ehrgott M. Multicriteria optimization. 2nd ed. Berlin, Heidelberg: Springer; 2010. https://doi.org/10.1007/3-540-27659-9.
[45] IBM-software. CPLEX Callable Library (C API) Reference Manual. IBM Doc 2021. https://www.ibm.com/docs/en/cofz/12.8.0?topic=zos-cplex-callable-library-api-reference-manual (accessed December 23, 2023).
[46] Quinn M. Parallel programming in C with MPI and OpenMP. 1st ed. Tata McGraw Hill Higher Education; 2003.