mathematics

MDPI

Article

Exact and Heuristic Approaches for Weighted Throughput Maximization in Single-Machine Scheduling

Yi-Feng Hung *, Jia-Shain Lin, Chih-Hao Lai and Chia-Hui Chien

Department of Industrial Engineering and Engineering Management, National Tsing Hua University, Hsinchu 300044, Taiwan

* Correspondence: yifeng@ie.nthu.edu.tw

Abstract

This study investigates a single-machine throughput maximization problem (TMP), in which each job is characterized by a processing time, ready date, due date, and weight, and must be completed within its time window to yield profit. The objective is to select and schedule a subset of jobs to maximize the total weight of completed jobs. While existing studies primarily focus on special cases or heuristic approaches, this paper develops a unified exact solution framework for the general TMP. The main contribution is a job-sorting-based approach in which dominance rules are derived from structural properties induced by a partial ordering, while a complete ordering based on time-window midpoints is used to guide job sequencing. Based on this framework, several solution approaches are developed, including implicit enumeration (IE) methods, branch-and-bound (B&B) methods enhanced with linear programming (LP) relaxation, and a mixed-integer programming (MIP) model. A heuristic method is also proposed to generate high-quality initial solutions that accelerate exact methods. Computational experiments are conducted on a comprehensive set of instances. The results demonstrate that the proposed job sorting and dominance rules significantly improve computational efficiency, leading to substantial reductions in solution time compared with baseline methods.

Keywords: throughput maximization problem; production scheduling; mixed-integer programming; implicit enumeration method; branch-and-bound method; single-machine; weighted jobs

MSC: 90-08; 90-10

1. Introduction

Scheduling plays a critical role in production and operations management, where limited resources must be allocated efficiently to competing tasks. In many real-world environments, decision makers must determine not only how to schedule jobs, but also which jobs to select for processing in order to maximize overall profit or system performance. This leads to the throughput maximization problem (TMP), also known as the order selection problem, in which a subset of jobs is chosen and scheduled within given time constraints.

This study considers the single-machine TMP in which each job is characterized by a ready date, due date, processing time, and weight. A job must be fully processed within its time window to earn its associated payoff; otherwise, it is rejected. The objective is to maximize the total weight of completed jobs. Such problems arise in various production

Check for updates

Academic Editor: Chawis Boonmee

Received: 18 March 2026
Revised: 14 April 2026
Accepted: 17 April 2026
Published: 18 April 2026

Copyright: © 2026 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license.

settings, particularly in make-to-order environments where capacity is limited and jobs differ in profitability and timing requirements.

Although real-world production systems often involve multiple machines, single-machine models remain valuable for tactical capacity planning and bottleneck scheduling. In long-term planning, machines with similar functionalities are frequently aggregated into a single capacity resource to evaluate feasibility [1]. Furthermore, many small and medium-sized companies are centered around a dominant bottleneck machine whose capacity largely determines overall system performance. In such cases, single-machine scheduling models provide a robust framework for analyzing capacity feasibility and developing efficient scheduling algorithms. Recent survey studies, such as Martinelli et al. [2] and Ying et al. [3], further highlight the continued relevance and ongoing development of single-machine production scheduling in theoretical research and real-world applications.

In real-world settings, manufacturers often face the critical task of maximizing profit by selecting a subset of orders from a candidate pool such that the chosen orders can be feasibly scheduled. In this study, the terms "order" and "job" are used interchangeably. A solution to this problem must specify both the subset of jobs to be processed and their corresponding production schedule. Such scenarios commonly arise in production environments such as apparel manufacturing, where orders have distinct selling seasons and material lead times. Additionally, due to high setup costs and time constraints, splitting an order across multiple production periods is impractical; therefore, preemption is prohibited. Moreover, partial fulfillment is not permitted, as an order must be either fully completed or entirely rejected. Since some garments feature high-fashion designs while others target price-sensitive markets, the potential profit varies substantially across different orders.

Despite extensive research on TMP and related scheduling problems, most existing studies focus on special cases, extended models with additional constraints, or heuristic approaches for large-scale instances. Comparatively fewer studies address the general TMP with time windows and job selection decisions using efficient exact methods.

To address this gap, this paper develops several solution approaches, including an MIP model, a partial-ordering-based ($\Rightarrow$) heuristic sorting method, an IE approach with dominance rules, and a B&B method enhanced by LP-relaxation. Computational experiments are conducted to evaluate the effectiveness and efficiency of the proposed methods. The following example illustrates the constraints and objectives of the TMP.

Illustration Example

There are six jobs in this example. The ready date and the due date (time window) of these jobs are shown in Figure 1. The optimal set of selected jobs is {1, 2, 4, 5, 6} and an optimal schedule is shown as a Gantt Chart on the time axis at the bottom of the figure. There are only three cases of time window overlap; they are between Jobs 1 and 2, between Jobs 3 and 4, and between Jobs 5 and 6; hence, this problem can be treated as three independent problems. Observing the figure, moving the processing of Job 1 later will make the time insufficient for processing Job 2. Hence, Job 1 should start at r_1 and Job 2 is processed right after Job 1 is completed. That $w_3 < w_4$ is assumed in this example. The ready date of Job 3 is earlier than that of Job 4. However, scheduling Job 3 will make the more profitable Job 4 impossible to be produced. Consequently, one should make the machine idle between interval $[r_3, r_4]$ and schedule Job 4 starting at r_4. Assume $w_5 < w_6$. Jobs 5 and 6 have an identical ready date, selecting the more profitable Job 6 to start at its ready time makes Job 5 impossible to be scheduled. However, processing Job 5 before Job 6 will make the production of both jobs feasible. The above solutions to the three subproblems are not only feasible but also optimal.

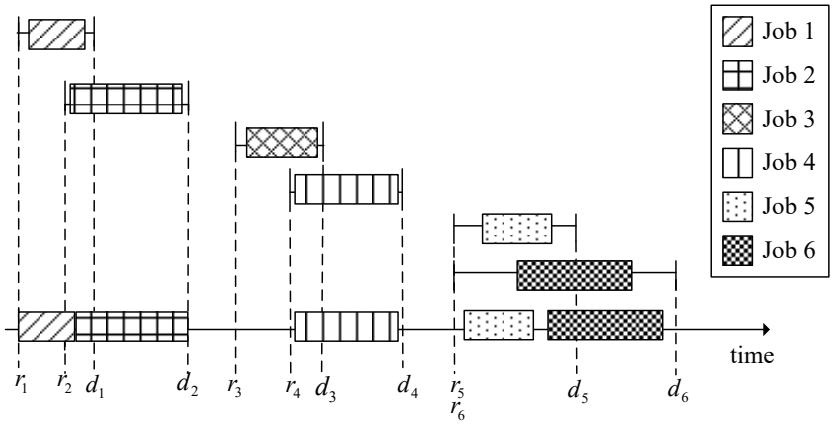


Figure 1. An illustrative example.

The remainder of this paper is organized as follows. Section 2 reviews the relevant literature. Section 3 presents the problem formulation. Section 4 describes the proposed solution approaches. Sections 5 and 6 report computational experiments and results, respectively. Section 7 concludes the paper.

2. Literature Review

2.1. Classical Throughput Maximization Problem

The TMP is a scheduling problem that aims to maximize the total weight of selected jobs. Several studies have investigated this problem under various assumptions regarding job attributes. Karp [4] proved that a special case, in which all jobs have identical ready dates ($r_j = r$), is NP-hard. Subsequently, Sahni [5] proposed a dynamic programming (DP) approach to solve this special case. In recent years, research on scheduling has also explored emerging methodologies, including artificial intelligence–based approaches. For example, Bouška et al. [6] and Liu et al. [7] investigated the application of learning-based techniques to scheduling problems.

2.2. Variants Based on Job Attributes

Various extensions of TMP have been studied by modifying job-related attributes, such as processing times, penalties, and resource requirements. Considering a variation where job weights are reduced by lateness penalties, Slotnick and Morton [8] developed a B&B method to identify optimal solutions, along with two heuristic methods for generating approximate solutions. Ghosh [9] subsequently proved that the problem studied by Slotnick and Morton is NP-hard. Building on these results, Slotnick and Morton [10] and Juraszek [11] extended the model by replacing the lateness penalty with a tardiness penalty. Slotnick and Morton [10] formulated the problem as an integer program and proposed a B&B method that incorporates LP-relaxation for bounding, in addition to heuristic approaches. Juraszek and Malgorzata [11] addressed the same problem in a multi-machine environment using an MIP model. They proposed a B&B algorithm, a simulated annealing (SA) heuristic, and a list scheduling algorithm to generate initial solutions for both the B&B and SA methods.

In the TMP, a job is considered eligible only if its processing time p_j is no longer than its time window R_j. The portion of a time window not occupied by processing is defined as slack time. Arkin and Silverberg [12] examined a special case in which no slack time exists for any job under a multi-machine setting and proposed a DP model for the problem. Later, Calinescu et al. [13] studied the same problem in a single-machine environment, introducing a DP algorithm suitable for large-scale problems, along with a randomized rounding algorithm and a list-based algorithm for small-scale instances.

If all jobs must be selected and processed, the TMP becomes equivalent to the problem of minimizing the weighted number of tardy jobs [14]. Several researchers have investigated this related problem in previous studies. Lawler and Moore [15] first examined a special case in which all jobs have identical ready dates. Subsequently, Lawler [16] considered a case where job processing times and weights are ordered in opposite directions. Kise et al. [17] analyzed a special case under specific assumptions on ready and due dates, where the order of ready dates matches the order of due dates for all jobs. They also proposed a DP algorithm to solve this case. In another special case, Lawler [18] developed a DP algorithm for problems in which all job weights are identical and preemption is permitted.

Several studies have focused on special cases regarding job processing times. Baptiste [19] considered a scenario in which all jobs have identical processing times and proposed a DP algorithm with a time complexity of $O(n^7)$, where n is the number of jobs, under the assumption that preemption is not permitted. For the preemptive case, the problem was shown to be solvable in $O(n^{10})$ time. Subsequently, Baptiste et al. [20] improved these results by presenting an algorithm with a reduced time complexity of $O(n^4)$ for the preemptive case originally studied by Baptiste [19].

A special case in which a job can be completed in a shorter duration by increasing resource usage at a higher cost is defined as the controllable processing time problem [21]. Yang and Geunes [14] investigated the single-machine TMP under the assumption of controllable processing times, where a tardiness cost is incurred if a job is completed after its due date. The TMP that incorporates both controllable processing times and tardiness penalties is known as the job selection with controllable processing times and tardiness scheduling problem. Yang and Geunes also proposed heuristic approaches to solve this problem.

In modern high-tech manufacturing environments, the profit obtained from completing a job may vary over time. Many products, such as memory chips and computers, have short life cycles and rapidly declining selling prices. To account for decreasing profits, Yang [22] assumed that the profit generated by a job decreases exponentially with its completion time and developed both a B&B algorithm and a heuristic method to address this problem. Furthermore, addressing scenarios with time-sensitive revenue and operational complexities, Chaurasia and Singh [23] investigated the single-machine problem with release dates and sequence-dependent setup times. Such setup times commonly occur when different jobs require distinct setup procedures. To solve their model, they proposed a hybrid steady-state genetic algorithm and an evolutionary algorithm with guided mutation.

Silva et al. [24] studied scheduling problems involving processing times, release dates, due dates, deadlines, revenues, and penalties. They proposed two exact algorithms: one based on Lagrangian relaxation and the other based on column generation. Both methods effectively handle sequence-dependent setup times and complex job constraints. Li and Ventura [25] also focused on single-machine order acceptance and scheduling (OAS) problems with revenues, due dates, and tardiness costs. They developed three exact algorithms: the first based on DP, the second employing a staged DP with gradual constraint recovery, and the third incorporating Lagrangian relaxation to enhance performance. Further exploring the single-machine OAS problem, de Weerdt et al. [26] addressed scheduling with release times, deadlines, setup times, and job rejection. They proposed an exact algorithm that is fixed-parameter tractable based on slack and width, where width refers to the number of overlapping time windows. Tarhan and Oğuz [27] introduced a metaheuristic approach for the OAS problem in a single-machine, make-to-order setting with release times and sequence-dependent setup times. Their method integrates a time-bucket-based MIP model with variable neighborhood search and tabu search.

Under the Just-in-Time (JIT) philosophy, jobs are often required to be completed exactly on their due dates to avoid both earliness and tardiness penalties. Gerstl and Mosheiov [28] addressed scheduling problems in such environments by proposing a DP algorithm to maximize the weighted number of JIT-compliant jobs for a given job sequence. These problems were shown to be solvable in polynomial time for both identical and unrelated machine settings.

More recently, Kaul et al. [29] studied a single-machine scheduling problem of time-window jobs with the objective of minimizing the number of tardy jobs.

2.3. Machine Environments of TMP

In addition to job-related variations, TMP has also been extended to different machine environments. By distinguishing between machine types (identical vs. unrelated) and job weight types (identical vs. arbitrary), Bar-Noy et al. [30] considered four variants of the TMP and proposed constant-factor approximation algorithms based on LP to solve each variant. They also presented a combinatorial algorithm to solve the TMP under the setting of arbitrary job weights and unrelated machines. Building on this work, Berman and Dasgupta [31] developed combinatorial algorithms that do not rely on LP and are significantly faster than the method proposed by Bar-Noy et al. [30]. Motivated by the work of Baptiste [19], Chuzhoy et al. [32] studied the TMP in both single-machine and multi-machine environments and proposed a DP algorithm. Zhang and Lu [33] addressed a parallel-machine scheduling problem that incorporates release dates and allows job rejection. They proposed a DP algorithm and a fully polynomial-time approximation scheme (FPTAS) to minimize the sum of the makespan and the total penalty incurred from rejected jobs.

Focusing on an unrelated parallel machine environment where all jobs are assumed to have a ready time of zero ($r_j = 0$), Wang and Ye [34] developed two MIP models and a formulation-based B&B algorithm for the OAS problem. Wang et al. [35] extended this work by considering additional complexities, such as machine-dependent setup times and maximum machine availability, to maximize total revenue while accounting for costs associated with the makespan. They proposed a two-layer logic-based Benders decomposition method, which decomposes the problem into an acceptance master problem and assignment-sequencing subproblems.

For the problem of minimizing makespan on unrelated parallel machines with sequence-dependent setup times, Fanjul-Peyró et al. [36] introduced a mathematical programming-based algorithm that employs two different MIP models (one for machine selection and the other for sequencing) while iteratively incorporating results as lower bounds. Addressing the OAS problem with time windows and sequence-dependent setup times on unrelated parallel machines, Mahmoudinazlou et al. [37] proposed a hybrid population-based heuristic that combines an Imperialist Competitive Algorithm with simulated annealing. Considering sequence-dependent setup times in an unrelated parallel-machine environment, Jos et al. [38] proposed three MIP models: one precedence-based, one position-based, and one time-indexed.

2.4. Online Throughput Maximization

Another important research direction considers online versions of the TMP, where job information is not known in advance. In real-world environments where jobs arrive sequentially and their attributes are unknown in advance, the scheduling problem becomes an online problem. This setting requires dynamically generating a sequence of decisions to assign resources to tasks in real time. Consequently, schedulers must often make decisions based solely on information available from jobs that have already arrived. Baruah et al. [39]

and Koren and Shasha [40] studied online scheduling problems with preemption, while Lipton and Tomkins [41] examined the non-preemptive version. Chin et al. [42] investigated an online scheduling problem arising from network buffer management. They assumed that all jobs have identical processing times, that ready dates and due dates are integers, and that weights are real numbers. A randomized algorithm was proposed to solve the problem under these assumptions. Liu et al. [43] deal with online and offline versions of the job selection or rejection problem.

2.5. Applications of TMP

Beyond theoretical developments, TMP has also been applied to various practical domains. Bar-Noy et al. [44] presented a general framework for solving the TMP and applied it to various domains, such as bandwidth allocation for sessions in communication networks and dynamic storage allocation. Subsequently, Bar-Noy et al. [45] considered the TMP with batching in a multi-machine environment, where jobs are grouped into several families. Each family is associated with a specific processing time, and jobs belonging to the same family can be batched and processed together on a single machine. The processing time of each batch is equal to the processing time assigned to its job family. The batch scheduling methods proposed by Bar-Noy et al. [45] have been applied to practical problems, including client scheduling in multimedia-on-demand systems and thermal treatment scheduling for multi-layer ceramic packaging.

Despite the extensive literature on TMP, most existing studies focus on special cases, extended models with additional constraints (e.g., sequence-dependent setup times, controllable processing times), or heuristic approaches for large-scale instances. Comparatively fewer studies address the general single-machine TMP with ready dates, due dates, and job selection decisions using efficient exact methods.

To the best of our knowledge, no existing study investigates the TMP under the exact set of assumptions considered in this paper. The works of Og et al. [46] and Mahmoudinazlou et al. [37] are the most closely related; however, their models incorporate sequence-dependent setup times, which are not considered in this study. Furthermore, while these studies primarily emphasize heuristic or metaheuristic approaches, the present study focuses on the development of exact optimization methods.

Specifically, this paper investigates several solution approaches, including a partial-ordering-based heuristic sorting method, an IE approach with dominance rules, and a B&B method enhanced by LP-relaxation and an MIP model. The proposed approaches provide both theoretical insights and practical solution techniques for solving the TMP efficiently under realistic time-window constraints.

3. Problem Description

We now formally define the problem. Let n be the total number of jobs, where each job j is characterized by four attributes: its ready date r_j, due date d_j, processing time p_j, and weight w_j. The time interval $[r_j, d_j]$, denoted as R_j, is defined as the time window of job j; the length of $d_j - r_j$ is denoted by $|R_j|$. A job must be processed within its time window $[r_j, d_j]$ without preemption, and no payoff is granted for tardy or partially completed jobs. Following the three-field notation of Graham et al. [47], the TMP considered in this study can be represented as $1/\ r_j,\ d_j\ /\sum_{j\in J} w_j I_j$. Here, $j \in J$ represents the set of all jobs, and I_j is a binary decision variable that equals 1 if job j is selected and completed within its time window, and 0 otherwise.

This study distinguishes between the terms solution and schedule. A solution specifies a subset of selected jobs and their processing sequence but does not define exact start times. In contrast, a schedule explicitly defines the start and end times for each selected job.

A job is feasibly scheduled if its start time is no earlier than its ready date (r_j) and its completion time is no later than its due date (d_j). Accordingly, a solution is deemed feasible if all selected jobs can be scheduled within their respective time windows in the specified sequence. Since the objective value, the total weight of selected jobs, is independent of specific start and end times, multiple feasible schedules may correspond to the same feasible solution. This occurs because job start and end times can often be shifted within their allowable windows without violating feasibility. An optimal solution is defined as a feasible solution whose total weight is at least as large as that of any other feasible solution. The focus of this study is to identify a single optimal solution rather than enumerating all alternative optima.

The degree of overlap among job time windows reflects the difficulty level of the scheduling problem. A TMP instance with no overlap between any pair of job time windows is the most trivial case. In such scenarios, selecting all jobs and scheduling each within its respective time window yields an optimal solution. In contrast, the most difficult case arises when all jobs share identical time windows, such that $r_j = r$ and $d_j = d$, $\forall j \in J$. This version of TMP is equivalent to the classical knapsack problem. By treating each job as an item, the common time window length as the knapsack capacity, the job processing time as the item weight, and the job weight as the item value, TMP can be transformed into a knapsack problem. The knapsack problem was proven to be NP-hard by Mathews [48]. Furthermore, Karp [4] demonstrated that TMP is NP-hard even when all jobs share the same ready date. Since both of these problems can be reduced to the TMP investigated in this study, the complexity of TMP ranges from a trivial case to a difficult NP-hard problem, depending on the degree of time window overlap.

The TMP can be formulated as the following optimization problem:

$$\text{Maximize } \sum_{j \in J} w_j I_j \tag{1}$$

subject to scheduling feasibility constraints that ensure:

1. Each selected job is processed within its time window $[r_j, d_j]$;
2. No two jobs overlap in processing on the single machine,
3. Processing is non-preemptive.

Here, $I_j \in \{0, 1\}$ indicates whether job j is selected and completed.

The following assumptions are adopted in this study:

1. Jobs are processed on a single machine;
2. Preemption is not allowed;
3. Each job must be completed within its time window to earn its weight;
4. The machine can process at most one job at a time;
5. All job parameters are known in advance.

4. Solution Approaches

Two major classes of approaches are commonly used to solve NP-hard scheduling problems. The first category consists of exact methods, which systematically explore the solution space to identify an optimal solution, typically through enumeration or mathematical optimization techniques. The second category includes heuristic methods, which aim to generate high-quality solutions within limited computation time.

This study focuses on the development of exact methods based on an enumeration framework. In particular, several solution approaches are proposed and compared, including an MIP model solved by Gurobi version 7.5, an IE method, and a B&B method enhanced with midpoint-based ordering, dominance rules, and LP-relaxation. A heuristic method is also developed to generate initial feasible solutions and support the exact approaches.

Almost all exact combinatorial optimization methods, including commercial integer programming solvers, rely on enumeration combined with techniques that reduce the size of the search tree and, consequently, the computational burden. The main contribution of this paper is the development of a job-sorting-based framework in which dominance rules are derived from structural properties induced by a partial ordering ($\Rightarrow$), while a complete ordering based on time-window midpoints is used to guide job sequencing. To evaluate this contribution, an enumerative method incorporating midpoint-based ordering and dominance rules is compared with a pure enumerative method. When these two enumerative approaches are further enhanced with LP relaxation at each explored node, they become B&B methods. In addition, a heuristic method that rapidly generates feasible solutions is used to provide improved lower bounds, thereby accelerating the B&B process. The heuristic solutions can also be supplied to the MIP solver as initial solutions to enhance its performance. Figure 2 illustrates the five solution approaches compared in this study, as well as the relationships among them and the development flow of the proposed techniques.

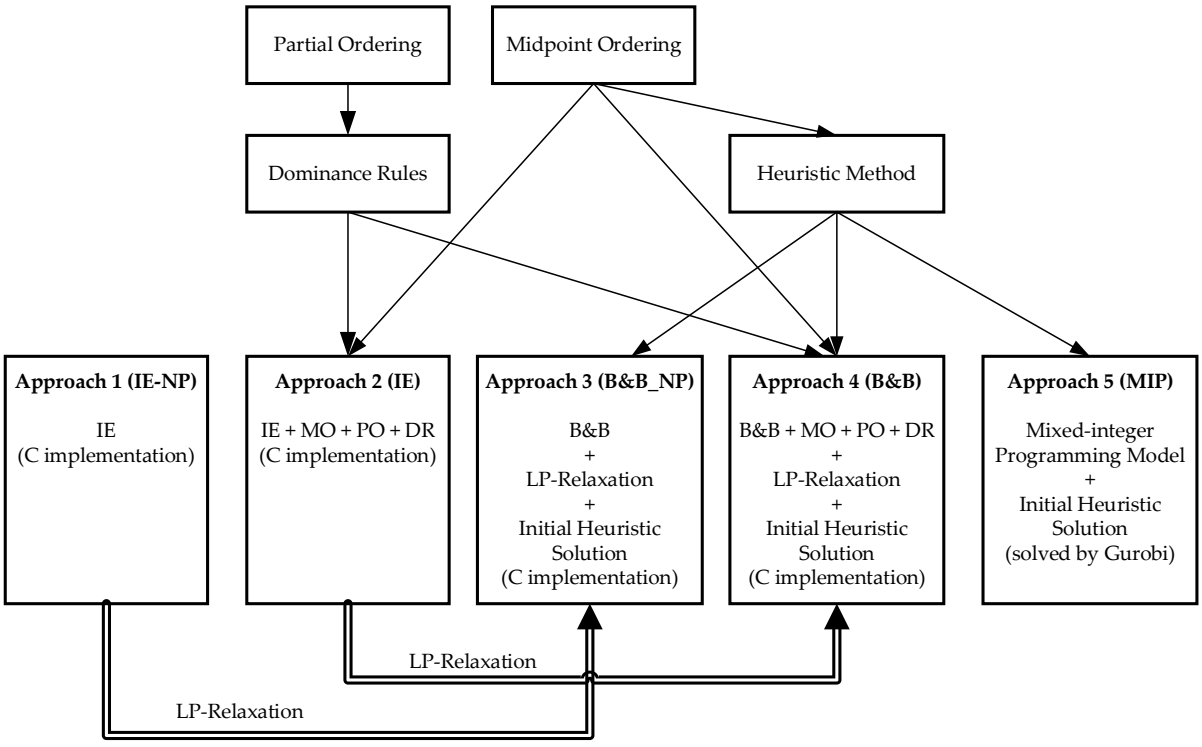


Figure 2. Relationships among the five solution approaches. The figure illustrates how midpoint-based ordering (MO), partial ordering ($\Rightarrow$) (PO), dominance rules (DR), and LP relaxation are incorporated into IE and B&B methods. The heuristic method provides initial feasible solutions to accelerate the exact solution approaches.

Section 4.1 introduces the heuristic method. Section 4.2 presents the MIP model. Section 4.3 details the IE method, which incorporates four dominance rules to prune the search tree and reduce computation time. Section 4.4 presents an LP relaxation formulation used to solve subproblems at nodes in the B&B enumeration tree.

4.1. Heuristic Method

The focus of this heuristic method is to prioritize jobs based on their payoff per unit of processing time. The weight rate of job j, denoted by ρ_j^W, is defined as

$$\rho_j^W = {w_j}\Big/{p_j} \tag{2}$$

where w_j is the weight and p_j is the processing time of job j. Hence, ρ_j^W represents the weight per unit of processing time.

The capacity demand rate of job j, denoted by $\rho_j^{CD}(t)$, is defined as a time-dependent function:

$$\rho_j^{CD}(t) = \begin{cases} {p_j}\Big/{d_j - r_j}, & \text{if } r_j \le t \le d_j \\ 0, & \text{otherwise} \end{cases}, \tag{3}$$

where ${p_j}\Big/{d_j - r_j}$ represents the average processing requirement per unit of time over the time window $[r_j, d_j]$ of job j.

The overall capacity demand rate at time t, aggregated across all jobs, is defined as:

$$\overline{\rho}^{CD}(t) = \sum_{j=1}^{n} \rho_j^{CD}(t), \tag{4}$$

which represents the total processing load at time t.

The heuristic prioritizes jobs based on their weight rates ρ_j^W. Specifically, all jobs are initially sorted in descending order of ρ_j^W, so that jobs with higher payoff per unit processing time are considered earlier for inclusion in the schedule.

Given a partially constructed solution, a candidate job can be inserted into several possible slots: before the first job, between any two consecutive jobs, or after the last job in the current sequence. To maximize the available space within a specific slot for potential insertion, jobs scheduled before the slot are shifted as early as possible, while those scheduled after the slot are shifted as late as possible. This is achieved by calculating the earliest completion times for the jobs preceding the slot and the latest start times for the jobs following the slot.

Let $[j]$ be the job at position j in the current sequence H, and let $r_{[j]}$, $p_{[j]}$, and $d_{[j]}$ be its ready date, processing time, and due date, respectively. Let $C_{[j]}^E$ and $S_{[j]}^L$ denote the earliest completion time and the latest start time of job $[j]$, respectively. Furthermore, let $n(H)$ be the number of jobs currently scheduled in H.

Let s denote the index of a slot. Slot s refers to the position before the s-th job in the current schedule. Specifically, slot 1 is located before the first job, while slot $n(H)+1$ corresponds to the position after the last scheduled job, where $n(H)$ represents the number of jobs currently in the sequence H. Consider job j for insertion into slot s.

For a candidate job j to be inserted into slot s, a feasible time window $R'_{j,s}$ is defined. This window represents the allowable interval in which job j can be scheduled, constrained by its ready date r_j, due date d_j, and the time availability between adjacent jobs in the current schedule. The interval $R'_{j,s}$ is computed as follows:

- If $s = 1$ (before the first job): $R'_{j,s} = \left[r_j, \min\left(S_{[s]}^L, d_j\right)\right]$
- If $1 < s < n(H)+1$ (between two jobs): $R'_{j,s} = \left[\max\left(C_{[s-1]}^E, r_j\right), \min\left(S_{[s]}^L, d_j\right)\right]$
- If $s = n(H)+1$ (after the last job): $R'_{j,s} = \left[\max\left(C_{[n(H)]}^E, r_j\right), d_j\right]$

The length of this time window is denoted as $\left|R'_{j,s}\right|$. If $\left|R'_{j,s}\right| < p_j$, where p_j is the processing time of job j, then job j cannot be inserted into slot s. If no slot has sufficient space, then job j is excluded from the current schedule, and the next job in the sorted list (based on weight rate) is evaluated.

To determine the best slot for a candidate job, the heuristic evaluates each feasible slot based on the average capacity requirement within its allowable window $R'_{j,s}$. The average capacity demand $\chi_{j,s}$ is calculated by integrating the total demand curve $\int_{R'_{j,s}} \overline{\rho}^{CD}(t)dt$ over the interval $R'_{j,s}$ and dividing the result by the window length $\left|R'_{j,s}\right|$:

$$\chi_{j,s} = \int_{R'_{j,s}} \overline{\rho}^{CD}(t)dt \Big/ \left|R'_{j,s}\right| \tag{5}$$

This metric reflects the average processing load within the potential insertion interval. The slot with the minimum average capacity demand $\chi_{j,s}$ is selected for inserting the job. This criterion identifies the slot that is the least congested in terms of competing job demands within its feasible time window.

4.2. MIP Model

To provide a benchmark and enable comparison with general-purpose optimization approaches, the TMP is formulated as an MIP model. The formulation captures both job selection and sequencing decisions within a unified framework. Unlike the proposed enumeration-based methods, which exploit problem-specific structural properties, the MIP model is solved using a commercial solver and serves as a baseline for evaluating solution quality and computational performance.

The MIP model is presented here in Formulation 1.

Formulation 1.

Sets and Indices

J: set of all jobs.

i, j: index of jobs, $i, j \in J$.

Decision variables

I_j: Binary variable, where $I_j = 1$ if job j is selected and 0 otherwise, $\forall j \in J$.

Y_{ij}: Binary variable for $\forall i, j \in$ J and $i < j$. When both jobs i and j are selected, $Y_{ij} = 1$ indicates that job i is scheduled before job j, while $Y_{ij} = 0$ indicates that job j is scheduled before job i.

S_j: Continuous variable denoting the start time of job $j, \forall j \in J$. The completion time of job j is thus defined as $S_j + p_j$.

M: A sufficiently large positive constant.

Objective Function

$$\text{Maximize } \sum_{j \in J} w_j I_j \tag{6}$$

Subject to

$$r_j \leq S_j, \ \forall j \in J \tag{7}$$

$$S_j + p_j \leq d_j, \ \forall j \in J \tag{8}$$

$$S_j \geq (S_i + p_i) - M(1 - Y_{ij}) - M(1 - I_i) - M(1 - I_j), \ \forall i, j \in J, \ i < j \tag{9}$$

$$S_i \geq (S_j + p_j) - MY_{ij} - M(1 - I_i) - M(1 - I_j), \ \forall i, j \in J, \ i < j \tag{10}$$

$$Y_{ij} \in \{0, 1\}, \ \forall i, j \in J, \ i < j \tag{11}$$

$$I_j \in \{0, 1\}, \ \forall j \in J \tag{12}$$

Formulation 1 presents a MIP model for the considered scheduling problem. The objective function (6) aims to maximize the total weight of the selected jobs. Constraint (7)

ensures that the start time of each job is no earlier than its ready date. Constraint (8) enforces that the completion time of each job does not exceed its due date. Constraints (9) and (10) manage the precedence relationships between any two selected jobs; specifically, if two jobs are selected, one must precede the other. Constraints (11) and (12) specify that Y_{ij} and I_j are binary variables, respectively. In Formulation 1, the number of continuous variables S_j and binary variables I_j is n each, while the number of binary variables Y_{ij} is $\frac{n(n-1)}{2}$. Solving this MIP yields an optimal solution to the TMP. The heuristic method introduced in Section 4.1 can be used to generate an initial feasible solution to facilitate the MIP solving process. This MIP model is solved by Gurobi [49] in the experiment results presented in Section 6.

4.3. IE Approach

To solve the TMP optimally using problem-specific structural properties, an IE method is developed. Unlike the MIP model, which relies on a general-purpose solver, the IE method explicitly explores the solution space by constructing feasible job sequences. Each node in the enumeration tree represents a partial sequence of selected jobs, and the method systematically extends these sequences while checking feasibility with respect to time window constraints. This framework serves as the basis for incorporating additional techniques, including midpoint-based ordering for guiding the search and dominance rules derived from the partial ordering ($\Rightarrow$) to prune the search tree, thereby improving computational efficiency.

This section presents an IE procedure using a depth-first search (DFS) scheme to identify an optimal solution. Let l denote the level in the enumeration tree. A node at level l represents a partial solution consisting of l scheduled jobs. The sequence of these l jobs is defined by the path from the root node to the current node. By appending a not-yet-selected job to the current partial solution, a new sequence is generated, which corresponds to a child node at level $l + 1$ in the tree.

Suppose there are n jobs in a TMP instance. The root node at level 0 represents an empty solution with no selected jobs. From the root, n branches can be created, each corresponding to the selection of one of the n jobs, leading to child nodes at level 1. In general, a node at level l branches into $n - l$ child nodes at level $l + 1$, each representing the addition of a not-yet-selected job. Therefore, the number of nodes at level l is given by the permutation $P_l^n = \frac{n!}{(n-l)!} = n \times (n-1) \times \ldots \times (n-l+1)$. At the bottom level (level n), there are $n!$ nodes, each representing a complete sequence of all n jobs. Consequently, the total number of nodes in a complete enumeration tree is: $P_0^n + P_1^n + \cdots + P_{n-1}^n + P_n^n$.

By developing heuristic sorting, dominance rules, and an LP relaxation of subproblems, each discussed in detail below, this study effectively reduces the number of nodes generated in the enumeration tree, thereby significantly decreasing computation time.

Let H denote the solution associated with the current node in the enumeration tree. Let $\omega(H)$ represent the set of jobs included in H, $\lambda(H)$ denote the last job in H, and $C^E_{\lambda(H)}$ be the earliest completion time of job $\lambda(H)$. The notation $H + j$ refers to the solution obtained by appending a job $j \in J - \omega(H)$ immediately after the last job in H.

Let $P\{\tau|J\}$ denote a TMP instance in which the set of considered jobs J can only be scheduled after time τ. Let $P^*\{\tau|J\}$ be an optimal solution to $P\{\tau|J\}$, and let $O^*\{\tau|J\}$ denote the corresponding optimal objective value of $P^*\{\tau|J\}$.

Lemma 1. *For a given job set J and two time points τ_1 and τ_2 , consider the problems $P\{\tau_1|J\}$ and $P\{\tau_2|J\}$, with corresponding optimal objective values $O^*\{\tau_1|J\}$ and $O^*\{\tau_2|J\}$, respectively. If $\tau_1 < \tau_2$, then $O^*\{\tau_1|J\} \geq O^*\{\tau_2|J\}$.*

Proof. Since $\tau_1 \leq \tau_2$ and the set of selectable jobs is identical for the two problems, any optimal solution of $P\{\tau_2|J\}$ is also feasible for $P\{\tau_1|J\}$. In a maximization problem, the

objective value of a feasible solution cannot exceed that of an optimal solution. Therefore, the objective value of a feasible solution in a maximization problem is less than or equal to that of an optimal one. Therefore, $O^*\{\tau_1|J\} \geq O^*\{\tau_2|J\}$. □

Lemma 1 indicates that an earlier available starting time provides greater flexibility for scheduling jobs, and therefore cannot lead to a worse optimal objective value under the same job set.

Lemma 2. *For a fixed time τ and two job sets J_1 and J_2, consider the problems $P\{\tau|J_1\}$ and $P\{\tau|J_2\}$, with corresponding optimal objective values $O^*\{\tau|J_1\}$ and $O^*\{\tau|J_2\}$, respectively. If $J_1 \supset J_2$, then $O^*\{\tau|J_1\} \geq O^*\{\tau|J_2\}$.*

Proof. Since $J_1 \supset J_2$ and the time horizon is identical for both problems, any optimal solution of $P\{\tau|J_2\}$ is also feasible for $P\{\tau|J_1\}$. In a maximization problem, the objective value of a feasible solution cannot exceed that of an optimal solution. Therefore, $O^*\{\tau|J_1\} \geq O^*\{\tau|J_2\}$. □

Lemma 2 indicates that enlarging the set of available jobs cannot reduce the optimal objective value. With more candidate jobs to choose from, the decision maker has greater flexibility, and therefore the optimal objective value cannot be worse than that obtained from a subset of jobs. Together, Lemmas 1 and 2 show that increasing scheduling flexibility, either through earlier start times or larger job sets, cannot reduce the optimal objective value.

Based on the assumption that a selected job must be fully processed within its time window, Dominance Rule 1 follows directly.

Dominance rule 1. Consider the subproblem $P\left\{C^E_{\lambda(H)}\middle|J-\omega(H)\right\}$ For any job $j \in J-\omega(H)$, if $\max\left(r_j, C^E_{\lambda(H)}\right)+p_j > d_j$, then the child node $H+j$ can be fathomed. □

This rule eliminates jobs that cannot be feasibly scheduled after the current partial sequence. If the earliest possible start time of job j, given by $\max\left(r_j, C^E_{\lambda(H)}\right)$, already leads to a completion time beyond its due date, then job j cannot be included in any feasible extension of the current solution. Therefore, such nodes can be safely pruned without losing optimality.

4.3.1. Methods Based on Ordering Relations

This subsection utilizes two ordering relations: a partial ordering $\Rightarrow$ for deriving dominance rules, and a midpoint-based ordering $\rightarrow$ for guiding job sequencing.

Definition 1. *(Relation operator $\Rightarrow$) For two jobs $i, j \in J$, the relation $i \Rightarrow j$ holds if and only if $r_i \leq r_j$ and $d_i \leq d_j$.*

According to this definition, jobs $i \Rightarrow j$ means that the time window of job i is no later than that of job j; that is, both the ready date and the due date of job i are less than or equal to those of job j.

Definition 2. *(Equivalence under relation $\Rightarrow$) For two jobs $i, j \in J$, they are considered equivalent under the relation $\Rightarrow$ if and only if $r_i = r_j$ and $d_i = d_j$.*

This definition identifies jobs with identical time windows. Such jobs are indistinguishable with respect to the relation $\Rightarrow$, and therefore can be treated equivalently when sorting the job set. Such jobs have identical scheduling flexibility and do not impose any ordering preference under the relation $\Rightarrow$.

Definition 3. *(Equivalence notation $\Leftrightarrow$) If two jobs $i, j \in J$ are equivalent under the relation $\Rightarrow$, they are denoted as $i \Leftrightarrow j$.*

Lemma 3. *For any two jobs $i, j \in J$, the relation $i \Leftrightarrow j$ holds if and only if both $i \Rightarrow j$ and $j \Rightarrow i$.*

Proof. (if) Suppose $i \Rightarrow j$ and $j \Rightarrow i$, then $r_i \leq r_j$, $d_i \leq d_j$, $r_j \leq r_i$, and $d_j \leq d_i$. Hence, we have $r_i = r_j$ and $d_i = d_j$, which implies jobs i and j are equivalent under $\Rightarrow$, i.e., $i \Leftrightarrow j$.

(only if) If $i \Leftrightarrow j$, then by Definition 2, $r_i = r_j$ and $d_i = d_j$. Therefore, both $i \Rightarrow j$ and $j \Rightarrow i$ hold. □

This lemma shows that the equivalence relation $\Leftrightarrow$ corresponds to a mutual relation under $\Rightarrow$.

Lemma 4. *The relation $\Rightarrow$ defined on the set J is a partial ordering.*

Proof. A partial ordering has to satisfy reflexivity, transitivity, and antisymmetry. reflexivity: For any job $i \in J$, since $r_i \leq r_i$ and $d_i \leq d_i$, we have $i \Rightarrow i$. transitivity: For any jobs i, j, $k \in J$, if $i \Rightarrow j$ and $j \Rightarrow k$, then $r_i \leq r_j \leq r_k$ and $d_i \leq d_j \leq d_k$. Hence, $r_i \leq r_k$ and $d_i \leq d_k$, which implies $i \Rightarrow k$. anti-symmetry: For any jobs i, $j \in J$, if $i \Rightarrow j$ and $j \Rightarrow i$, then $r_i = r_j$ and $d_i = d_j$. Therefore, i and j are equivalent under $\Rightarrow$; i.e., $i \Leftrightarrow j$. □

The job set J, together with the relation $\Rightarrow$, forms a partially ordered set (or poset) denoted as $(J, \Rightarrow)$. Two jobs $i, j \in J$ are called incomparable if neither $i \Rightarrow j$ nor $j \Rightarrow i$ holds. This occurs when one job has an earlier ready date but a later due date than the other, i.e., when ($r_i \leq r_j$ and $d_i \geq d_j$) or ($r_i \geq r_j$ and $d_i \leq d_j$). In such cases, neither job dominates the other under this relation. A poset becomes a total ordering if no pair of elements is incomparable. However, in the problem considered in this study, incomparable job pairs may exist. Therefore, the poset $(J, \Rightarrow)$ is not a total ordering. This partial ordering is used to support the development of dominance rules.

Definition 4. *(Relation operator $\rightarrow$) For two jobs $i, j \in J$, the relation $i \rightarrow j$ holds if and only if $r_i + d_i \leq r_j + d_j$.*

This condition is equivalent to comparing the midpoints of the time windows. Specifically, it implies that $\frac{r_i+d_i}{2} \leq \frac{r_j+d_j}{2}$, i.e., the midpoint of the time window of job i is less than or equal to that of job j.

Definition 5. *(Equivalence under relation $\rightarrow$) Two jobs $i, j \in J$ are considered equivalent under the relation $\rightarrow$ if and only if $r_i + d_i = r_j + d_j$.*

This condition is equivalent to $\frac{r_i+d_i}{2} = \frac{r_j+d_j}{2}$, i.e., the midpoints of the time windows of jobs i and j are identical. Such jobs are indistinguishable under the ordering defined by $\rightarrow$ and may be treated equivalently during sorting.

Definition 6. *(Equivalence notation $\leftrightarrow$) If two jobs $i, j \in J$ are equivalent under the relation $\rightarrow$, they are denoted by $i \leftrightarrow j$.*

This notation indicates that the two jobs have identical midpoint values of their time windows.

Lemma 5. *For two jobs $i, j \in J$, the relation $i \leftrightarrow j$ holds if and only if both $i \to j$ and $j \to i$ hold.*

Proof. (if) Suppose $i \leftrightarrow j$. Then, by Definition 5, $r_i + d_i = r_j + d_j$. This implies $r_i + d_i \leq r_j + d_j$ and $r_j + d_j \leq r_i + d_i$, and therefore both $i \to j$ and $j \to i$ hold.

(only if) Suppose $i \to j$ and $j \to i$. Then $r_i + d_i \leq r_j + d_j$ and $r_j + d_j \leq r_i + d_i$. These two inequalities imply $r_i + d_i = r_j + d_j$. Hence, $i \leftrightarrow j$. □

This lemma shows that equivalence under $\to$ corresponds to mutual ordering based on midpoint comparison.

Lemma 6. *The relation $\to$ defined on the set J is a total preorder.*

Proof. The relation $\to$ satisfies the following properties: Reflexivity: For any job $i \in J$, $r_i + d_i \leq r_i + d_i$, hence $i \to i$. Transitivity: For jobs $i, j, k \in J$, if $i \to j$ and $j \to k$, then $r_i + d_i \leq r_j + d_j$ and $r_j + d_j \leq r_k + d_k$. Adding these two inequalities yields $r_i + d_i \leq r_k + d_k$, which implies $i \to k$. Totality: For any two jobs $i, j \in J$, either $r_i + d_i \leq r_j + d_j$ or $r_j + d_j \leq r_i + d_i$, hence either $i \to j$ or $j \to i$ (or both). □

The job set J, together with the relation $\to$, forms a totally preordered set. Two jobs may be equivalent under this relation if they share the same midpoint. By grouping equivalent jobs into equivalence classes, the induced ordering among these classes becomes a total ordering.

Lemma 7. *For any two jobs $i, j \in J$, if $i \Rightarrow j$, then $i \to j$.*

Proof. Since $i \Rightarrow j$, we have $r_i \leq r_j$ and $d_i \leq d_j$. Adding these two inequalities yields $r_i + d_i \leq r_j + d_j$. Therefore, $i \to j$. □

This lemma shows that the partial ordering $\Rightarrow$ is consistent with the midpoint-based ordering $\to$.

Lemma 8. *Suppose two jobs $i, j \in J$ satisfy $i \leftrightarrow j$. Then, for any job $k \in J - \{i, j\}$, the following two statements hold.*

(1) If $k \to i$, then $k \to j$.
(2) If $i \to k$, then $j \to k$.

Proof. Since $i \leftrightarrow j$, we have $r_i + d_i = r_j + d_j$.

(1) If $k \to i$, then we have $r_k + d_k \leq r_i + d_i$.
Substituting $r_j + d_j$ for $r_i + d_i$, we obtain $r_k + d_k \leq r_j + d_j$, which implies $k \to j$.

(2) If $i \to k$, then we have $r_i + d_i \leq r_k + d_k$.
Substituting $r_j + d_j$ for $r_i + d_i$, we obtain $r_j + d_j \leq r_k + d_k$, which implies $j \to k$. □

This lemma shows that jobs with identical midpoints are interchangeable under the relation $\to$.

Lemma 9. *If no pair of jobs in J satisfies the equivalence relation $\leftrightarrow$, then there exists a unique sequence of jobs sorted by the relation $\to$. Furthermore, if there exists a subset $J^s \subset J$ such that every pair of jobs in J^s satisfies the relation $\leftrightarrow$, then the jobs in J^s appear consecutively as a group in the sequence sorted by $\to$.*

Proof. The result follows from Lemma 8 and the transitivity and totality properties of the relation $\rightarrow$. If no pair of jobs is equivalent under $\leftrightarrow$, then all jobs are strictly ordered by their midpoint values, yielding a unique sorted sequence. If a subset J^s consists of jobs equivalent under $\leftrightarrow$, then by Lemma 8 these jobs are interchangeable under the relation $\rightarrow$, and therefore they must appear consecutively in any sequence sorted by $\rightarrow$. □

Under the condition that two jobs $i, j \in J$ satisfy $i \rightarrow j$, there are ten possible cases for the relationships among r_i, d_i, r_j, and d_j. These cases are detailed in Appendix A. Among them, Cases 1 through 6 also satisfy $i \Rightarrow j$, with Case 4 further satisfying $i \Leftrightarrow j$. In addition, Cases 8 and 10 satisfy the equivalence condition $i \leftrightarrow j$.

Lemma 10. *If $i \rightarrow j$ and ($r_i \neq r_j$ or $d_i \neq d_j$), then $j \Rightarrow i$ cannot hold.*

Proof. Assume $i \rightarrow j$. Then $r_i + d_i \leq r_j + d_j$. Suppose, for contradiction, $j \Rightarrow i$ holds. Then, $r_j \leq r_i$ and $d_j \leq d_i$. Adding these inequalities gives $r_j + d_j \leq r_i + d_i$. Combining this with $r_i + d_i \leq r_j + d_j$, we obtain $r_i + d_i = r_j + d_j$.

Since $r_j \leq r_i$ and $d_j \leq d_i$, if either inequality were strict, then $r_j + d_j < r_i + d_i$, which is impossible. Therefore, $r_j = r_i$ and $d_j = d_i$. This contradicts the assumption that $r_i \neq r_j$ or $d_i \neq d_j$. Therefore, $j \Rightarrow i$ cannot hold. □

This lemma indicates that a strict ordering under the midpoint-based relation $\rightarrow$ prevents a reverse dominance relationship under $\Rightarrow$.

Lemma 11. *Consider two jobs $i, j \in J$ such that $i \Rightarrow j$ and $r_j < d_i$. Suppose H^{ji} is an optimal schedule in which job i is scheduled immediately after job j . Then, there exists a feasible schedule H^{ij} obtained by exchanging the positions of i and j, while keeping the positions of all other jobs in H^{ji} unchanged. Moreover, H^{ij} is also optimal.*

Proof. Let S_j^{ji}, C_j^{ji} and S_i^{ji}, C_i^{ji} denote the start and completion times of jobs j and i, respectively, in schedule H^{ji}. Let S_j^{ij}, C_j^{ij} and S_i^{ij}, C_i^{ij} denote the start and completion times of jobs j and i, respectively, in schedule H^{ij}. Since H^{ji} is feasible, $r_j \leq S_j^{ji}$, $r_i \leq S_i^{ji}$, $C_j^{ji} \leq d_j$, and $C_i^{ji} \leq d_i$. Construct schedule H^{ij} by swapping the positions of jobs i and j. Then, $S_i^{ij} = S_j^{ji}$, $C_j^{ij} = C_i^{ji}$.

(1) Feasibility of job i: Since $S_i^{ij} = S_j^{ji}$ and $r_j \leq S_j^{ji}$, we have $r_j \leq S_i^{ij}$. Because $i \Rightarrow j$, we have $r_i \leq r_j$, hence $r_i \leq S_i^{ij}$. Also, since job i is scheduled before job j in H^{ij}, its completion time satisfies $C_i^{ij} \leq C_i^{ji} \leq d_i$. Thus, job i remains feasible.

(2) Feasibility of job j: Since $C_j^{ij} = C_i^{ji} \leq d_i$ and $i \Rightarrow j$ implies $d_i \leq d_j$, we have $C_j^{ij} \leq d_j$. Also, because job j is scheduled after job i, its start time satisfies $S_j^{ij} \geq S_i^{ij} = S_j^{ji} \geq r_j$. Thus, job j remains feasible.

(3) Optimality: Since the set of selected jobs remains unchanged, the objective value is unchanged. Therefore, H^{ij} is also an optimal schedule. □

This lemma shows that when job i is no later and no looser than job j, scheduling i before j is always at least as good as the reverse order.

Lemma 12. *If two jobs i and j satisfy $i \Rightarrow j$, then any schedule in which job i is scheduled immediately after job j cannot be a unique optimal schedule.*

Proof. Consider two cases.

1. $d_i \leq r_j$: In this case, the time window $[r_i, d_i]$ of job i lies entirely before $[r_j, d_j]$. Therefore, it is impossible to schedule job j before job i. Hence, any schedule in which job i is scheduled immediately after job j is infeasible.
2. $d_i > r_j$: In this case, the time windows overlap. By Lemma 11, any optimal schedule in which job i is scheduled immediately after job j can be transformed into another optimal schedule by exchanging the position of i and j. Therefore, such a schedule cannot be unique. □

Therefore, when $i \Rightarrow j$, it is unnecessary to consider any schedule in which job i is scheduled immediately after j. Following Lemma 12, we establish the following dominance rule:

Dominance rule 2. If two jobs i and j satisfy $i \Rightarrow j$, then any child node obtained by scheduling job i immediately after job j in the current partial solution can be fathomed. □

This rule follows from Lemma 12, since any such node either corresponds to an infeasible schedule or can be transformed into another node with an equivalent or better solution by exchanging the positions of jobs i and j.

Lemma 13. *If every pair of jobs in J is comparable under the relation $\Rightarrow$, then there exists an optimal solution in which the sequence of selected jobs follows the nondecreasing order defined by $\Rightarrow$. Consequently, given this sorted order, at any node representing a selected job, it is sufficient to branch only to jobs that appear later in the sequence.*

Proof. Since every pair of jobs is comparable under $\Rightarrow$, the relation defines a total ordering on J. Consider any optimal solution. If the sequence of selected jobs violates this ordering, then there exists at least one adjacent pair of jobs j, i such that $i \Rightarrow j$ but j is scheduled before i. By Lemma 11, such a pair can be exchanged without violating feasibility or optimality. Repeatedly applying such exchanges transforms the sequence into one that follows the order defined by $\Rightarrow$ without changing the objective value. Hence, there exists an optimal solution consistent with the sorted order.

Therefore, when constructing solutions, it is sufficient to branch only to jobs that appear later in the sorted sequence. □

This lemma shows that, under a total ordering, the search space can be restricted to sequences consistent with the ordering without loss of optimality.

4.3.2. Heuristic Sorting of Jobs

As discussed in Lemma 13 and Dominance Rule 2, the relation $\Rightarrow$ offers a strong structural property for sequencing jobs when searching for an optimal solution. Specifically, if $i \Rightarrow j$, it is only necessary to consider scheduling job j after job i, rather than the reverse. However, the poset $(J, \Rightarrow)$ is not a total ordering; that is, not all pairs of jobs can be compared by $\Rightarrow$. As a result, it may not be possible to derive a complete sorting of all jobs based solely on $\Rightarrow$. In contrast, the relation $\rightarrow$ provides a complete ordering of jobs based on the midpoints of their time windows. Therefore, the relation $\Rightarrow$ is used for dominance-based pruning, while the relation $\rightarrow$ is used for heuristic ordering.

According to Lemma 10, sorting by $\rightarrow$ does not contradict the ordering defined by $\Rightarrow$ when a job pair is comparable under $\Rightarrow$. Among the ten possible cases describing the relationship between jobs i and j where $i \rightarrow j$ holds, six cases (Cases 1 through 6) also satisfy $i \Rightarrow j$. For these six cases, it is unnecessary to consider solutions in which job i is scheduled immediately after job j.

For a job pair i and j that satisfies $i \to j$ but not $i \Rightarrow j$, scheduling job i before job j is not guaranteed to yield an optimal solution. However, the relation $i \to j$ implies that $d_j - r_i \geq d_i - r_j$, which indicates that the interval $[r_i, d_j]$ (corresponding to scheduling job i before job j) is wider than the interval $[r_j, d_i]$ (which results from scheduling job j before job i). Therefore, when both jobs are selected under the condition $i \to j$, scheduling i before j provides more slack and flexibility than the reverse order. Consequently, sorting jobs according to the relation $\to$ offers a useful heuristic for guiding the search process.

Moreover, since a depth-first search (DFS) strategy is employed in the enumeration tree, following the branching sequence defined by the sorted list increases the likelihood of generating a high-quality solution early. If the first solution generated by the sorted sequence is feasible and not eliminated by any dominance rule, it serves as a strong initial incumbent solution in the search for optimality.

4.3.3. Additional Dominance Rules

In addition to the dominance rules presented in the previous subsection, this subsection introduces further rules to reduce branching in the enumerative search tree. These additional rules exploit the structural properties of the scheduling problem and the ordering relations defined in this study to identify and eliminate dominated or redundant branches during the search process.

Figure 3 illustrates that job i can be inserted between the last scheduled job in H and job j without delaying the earliest completion time of job j, denoted by C_j^E.

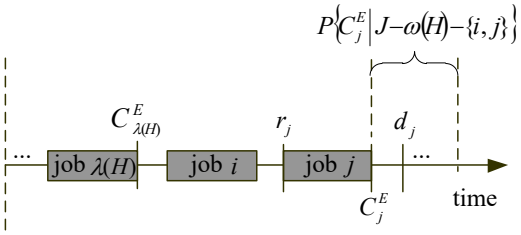


Figure 3. Job i can be inserted before job j without affecting the earliest completion time of job j.

Lemma 14. *Given a partial solution H, consider the subproblem $P\left\{C^E_{\lambda(H)} \middle| J-\omega(H)\right\}$. If there exist two jobs $i, j \in J-\omega(H)$ such that $\min(d_i, r_j) - \max\left(C^E_{\lambda(H)}, r_i\right) \geq p_i$, then the following inequality holds: $w_j + O^*\left\{C_j^E \middle| J-\omega(H+j)\right\} \leq w_i + w_j + O^*\left\{C_j^E \middle| J-\omega(H+i+j)\right\}$.*

Proof. The condition implies that job i can be inserted between the last scheduled job in H and job j without delaying the earliest completion time of job j, denoted by $C^E_{\lambda(H)}$ (see Figure 3). Consider two cases.

1. Job i is selected in an optimal solution of $P\left\{C_j^E \middle| J-\omega(H+j)\right\}$. In this case, job i is originally scheduled after job j without affecting C_j^E, we can exchange its position to occur before j. This increases the total weight accumulated before C_j^E by w_i, while potentially creating additional space after C_j^E for other jobs. Therefore, $O^*\left\{C_j^E \middle| J-\omega(H+j)\right\} \leq O^*\left\{C_j^E \middle| J-\omega(H+i+j)\right\}$.
2. Suppose job i is not selected in the optimal solution of $P\left\{C_j^E \middle| J-\omega(H+j)\right\}$. Since job i can be inserted before job j without affecting C_j^E, including i does not reduce the optimal value. Therefore, $O^*\left\{C_j^E \middle| J-\omega(H+j)\right\} = O^*\left\{C_j^E \middle| J-\omega(H+i+j)\right\}$

Combining both cases and noting that $w_j \leq w_i + w_j$, we obtain $w_j + O^*\left\{C_j^E \middle| J-\omega(H+j)\right\} \leq w_i + w_j + O^*\left\{C_j^E \middle| J-\omega(H+i+j)\right\}$. □

This lemma shows that inserting job i before job j, when feasible without delaying the earliest completion time of job j, cannot worsen the objective value and may improve it.

Dominance rule 3. Consider a subproblem $P\left\{C^E_{\lambda(H)} \middle| J-\omega(H)\right\}$ of a partial solution H. If there exist two jobs $i, j \in J-\omega(H)$ such that $\min(d_i, r_j) - \max\left(C^E_{\lambda(H)}, r_i\right) \geq p_i$, then the child node corresponding to the partial schedule $H+j$ can be fathomed. □

This rule follows from Lemma 14. Since inserting job i before job j does not delay the earliest completion time of job j and yields a solution that is at least as good as (and potentially better than) scheduling j immediately after H. Thus, branching directly from H to $H+j$ can be skipped.

Lemma 15. *Consider a subproblem* $P\left\{C^E_{\lambda(H)} \middle| J-\omega(H)\right\}$ *of a partial solution* H. *If there exist two jobs* $i, j \in J-\omega(H)$ *such that* $\left[\max\left(C^E_{\lambda(H)}, r_j\right)+p_j\right]+p_i > d_i$, $\left[\max\left(C^E_{\lambda(H)}, r_i\right)+p_i\right]+p_j > d_j$, $w_i \geq w_j$, *and* $\max\left(C^E_{\lambda(H)}, r_i\right)+p_i \leq \max\left(C^E_{\lambda(H)}, r_j\right)+p_j$, *then the following inequality holds:* $w_i + O^*\left\{C^E_i \middle| J-\omega(H+i)\right\} \geq w_j + O^*\left\{C^E_j \middle| J-\omega(H+j)\right\}$.

Proof. The two cases of adding job i or job j after H are illustrated in Figures 4 and 5, respectively.

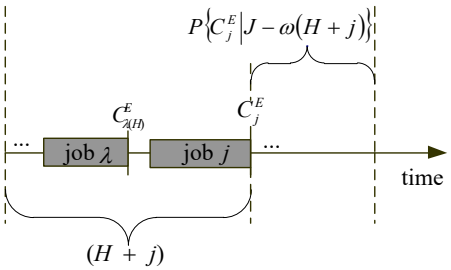


Figure 4. The problem $P\left\{C^E_j \middle| J-\omega(H+j)\right\}$ after solution $H+j$.

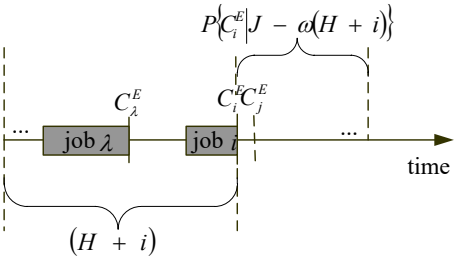


Figure 5. The Problem $P\left\{C^E_i \middle| J-\omega(H+i)\right\}$ after solution $H+i$.

The first two conditions imply mutual exclusion:

1. If job j is scheduled immediately after H, then job i cannot be scheduled afterward.
2. Similarly, if job i is scheduled after H, then job j cannot be scheduled afterward.

Thus, for both subproblems $P\left\{C^E_j \middle| J-\omega(H+j)\right\}$ and $P\left\{C^E_i \middle| J-\omega(H+i)\right\}$, the remaining job set is identical and equals $J-\omega(H)-\{i, j\}$. Let $C^E_i = \max\left(C^E_{\lambda(H)}, r_i\right)+p_i$, $C^E_j = \max\left(C^E_{\lambda(H)}, r_j\right)+p_j$. By assumption, $C^E_i \leq C^E_j$. By Lemma 1, this implies $O^*\left\{C^E_i \middle| J-\omega(H+i)\right\} \geq O^*\left\{C^E_j \middle| J-\omega(H+j)\right\}$. Since $w_i \geq w_j$, we obtain $w_i + O^*\left\{C^E_i \middle| J-\omega(H+i)\right\} \geq w_j + O^*\left\{C^E_j \middle| J-\omega(H+j)\right\}$. □

This lemma shows that when selecting one job that excludes the other, the job with the earlier completion time and no smaller weight dominates the alternative.

Following Lemma 15, we have Dominance rule 4. This rule follows from Lemma 15, since selecting job i excludes job j, and job i yields an objective value that is at least as good as that obtained by selecting job j.

Dominance rule 4. Consider a node representing solution H and its subproblem $P\left\{C^E_{\lambda(H)} \middle| J-\omega(H)\right\}$. If there exist two jobs $i, j \in J-\omega(H)$ such that $w_i \geq w_j$, $\left[\max\left(C^E_{\lambda(H)}, r_j\right)+p_j\right]+p_i > d_i$, $\left[\max\left(C^E_{\lambda(H)}, r_i\right)+p_i\right]+p_j > d_j$, and $\max\left(C^E_{\lambda(H)}, r_i\right)+p_i \leq \max\left(C^E_{\lambda(H)}, r_j\right)+p_j$, then the child node corresponding to solution $H+j$ can be fathomed. □

If only one of the two jobs i and j can be scheduled after partial solution H due to timing constraints, and job i has a weight no smaller than that of job j, and completes no later than job j, then selecting job i dominates selecting job j. Therefore, branching to $H+j$ is unnecessary. That is, any solution obtained from node $H+j$ cannot outperform a corresponding solution obtained from node $H+i$.

4.3.4. Implementation of the Enumerative Method

With the proposed dominance rules and heuristic job sorting, the enumerative method is implemented using a depth-first search (DFS) strategy. At each node, candidate jobs are explored according to the ordering defined by the relation $\rightarrow$, while the dominance rules are applied to prune infeasible or dominated branches.

The DFS strategy requires minimal memory, since only the nodes along the current path from the root to the active partial solution need to be stored. The search procedure is implemented as a recursive function, which is outlined in Appendix B.

4.4. B&B Method with LP-Relaxation

The B&B scheme presented in this section differs from the general-purpose framework used by Gurobi to solve the MIP model. In the proposed approach, each node in the branching tree represents a partial sequence of selected jobs, and branching corresponds to extending this sequence by selecting an additional eligible job. In contrast, Gurobi operates on the MIP formulation and explores nodes defined by restrictions on decision variables. Therefore, the proposed B&B method is a problem-specific enumeration scheme rather than the generic B&B procedure used by a commercial MIP solver.

Within this framework, an LP-relaxation is used to evaluate each node in the enumeration tree. Specifically, the LP-relaxation provides an upper bound for the corresponding subproblem. As discussed in Section 3, the problem under investigation is a non-preemptive scheduling problem, meaning that each job must be processed without interruption. The LP-relaxation relaxes this restriction by allowing jobs to be split across multiple processing periods. Consequently, portions of a job may be processed in non-consecutive time intervals in the relaxed solution. Under this relaxation, the original MIP model reduces to a linear program, whose optimal value serves as an upper bound for the node under consideration.

4.4.1. LP Relaxation Formulation of a Node at Level l

Let $R = \{r_1, r_2, \ldots, r_n\}$ be the set of ready dates and $D = \{d_1, d_2, \ldots, d_n\}$ be the set of due dates for all jobs. Define $B = R \cup D$ as the set of all ready and due dates, whose elements are referred to as break points. The LP-relaxation is constructed by discretizing the time horizon into time buckets defined by break points. For example, consider a job with processing time $p_j = 4$ and time window $[r_j, d_j] = [2, 10]$. In the LP-relaxation, this job may be partially processed in multiple time buckets within its time window, such as allocating 1 units in one interval and 3 units in another and separated by other jobs. This is in contrast to the original problem, where the job must be processed continuously without

interruption. The LP-relaxation model for the node corresponding to solution H at level l in the search tree is presented as follows.

Formulation 2.

Notations

t: Index of break points in B.

T': Total number of break points in B.

b_t: The t-th smallest element in B, such that $b_1 < b_2 < \ldots < b_{T'-1} < b_{T'}$. Time is partitioned into **time buckets** based on these break points. Specifically, time bucket t (for $t > 1$) corresponds to the interval $[b_{t-1}, b_t]$, while time bucket 1 corresponds to $[0, b_1]$.

α_j: The first time bucket in which job j can be processed.

β_j: The last time bucket in which job j can be processed.

$\omega(H)$: The set of jobs already included in the current solution H.

t': The time bucket containing the earliest completion time of the last scheduled job $\left(C^E_{[l]}\right)$, such that $b_{t'-1} \leq C^E_{[l]} < b_{t'}$.

J^*_t: The set of jobs eligible for processing in time bucket t, defined as $J^*_t = \{j | r_j \leq b_{t-1},\ b_t \leq d_j\}$.

A_t: The set of unscheduled jobs eligible in time bucket t, where $A_t = J^*_t \cap (J - \omega(H))$.

X_{jt}: A continuous decision variable representing the amount of time allocated to job j in time bucket t.

B^*_j: The set of time buckets in which job j can be processed, defined as $B^*_j = \{t | t \geq t',\ \alpha_j \leq t \leq \beta_j\}$.

Linear programming

$$\text{Maximize } \sum_{j \in J-\omega(H)} \left(w_j \times \frac{\sum_{t \in B^*_j} X_{jt}}{p_j} \right) + \sum_{j \in \omega(H)} w_j \tag{13}$$

Subject to

$$\sum_{j \epsilon A_{t'}} X_{jt'} \leq b_{t'} - C^E_{[l]} \tag{14}$$

$$\sum_{j \epsilon A_t} X_{jt} \leq b_t - b_{t-1},\ t = t'+1,\ t'+2,\ \ldots,\ T' \tag{15}$$

$$\sum_{t \epsilon B^*_j} X_{jt} \leq p_j,\ \forall j \in J - \omega(H) \tag{16}$$

The objective function (13) maximizes the total weight of jobs. In the LP relaxation, partially processed jobs contribute proportionally to their allocated processing time. Constraint (14) ensures that the total processing time assigned to unscheduled jobs in time bucket t' does not exceed the remaining available capacity in that interval. Constraint (15) ensures that, for each time bucket $t > t'$, the total assigned processing time does not exceed the duration of that bucket. Constraint (16) ensures that the total processing time assigned to any job does not exceed its required processing time p_j.

4.4.2. Feasibility Checks for the LP-Relaxation

Once the LP relaxation problem is solved, two conditions can be applied to verify whether the obtained solution is feasible for the original TMP.

Condition 1

In this study, each job must be either fully completed or entirely rejected. For a solution to be feasible, the total allocated processing time for each job j, calculated by $\sum_{t=\alpha_j}^{\beta_j} X_{jt}$, must be exactly either 0 or p_j. To reduce computation time, the verification procedure can be terminated early if any job j is found to satisfy $0 < \sum_{t=\alpha_j}^{\beta_j} X_{jt} < p_j$, indicating that the integrality condition is violated. To account for potential numerical round-off errors, our experiments apply the condition as $\delta < \sum_{t=\alpha_j}^{\beta_j} X_{jt} < p_j - \delta$, where δ represents a very small acceptable numerical tolerance.

Condition 2

In the solution to the LP relaxation problem, $X_{jt} > 0$ for a given job j and time bucket t indicates that X_{jt} units of processing time in bucket t are allocated to job j. For each job j, the

set of all nonzero X_{jt} values across different time buckets constitute its processing periods. For the solution to be feasible in the original non-preemptive problem, all processing periods of a job must be scheduled consecutively without interruption. In other words, the allocated time blocks for each job must form a contiguous interval.

Figure 6 illustrates such a feasible schedule, in which the processing time of each job forms an uninterrupted block. For example, in time bucket 2, the LP-relaxation solution indicates that processing time is allocated to both Jobs 1 and 2, but it does not specify their processing order. Similarly, in time bucket 4, the capacity is divided between jobs 3 and 2. By sequencing Job 1 before Job 3 in bucket 2, and Job 3 before Job 2 in bucket 4, the processing periods can be arranged so that Jobs 1, 3, and 2 are each processed continuously. Therefore, the solution shown in Figure 6 corresponds to a feasible schedule for the original problem.

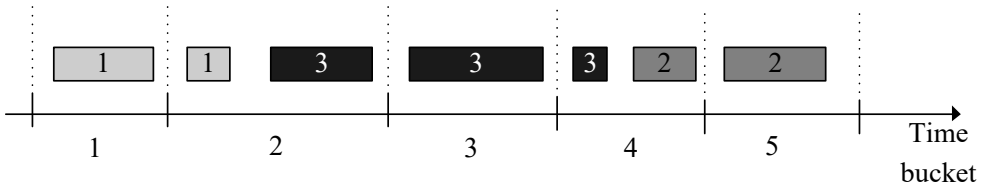


Figure 6. An LP relaxation solution that can be arranged into a feasible non-preemptive schedule.

A schedule in which at least one job is split into multiple non-consecutive processing periods is infeasible for the original problem. Figure 7 illustrates such an infeasible schedule. In this example, regardless of how the two processing periods of Jobs 2 and 3 in time bucket 3 are sequenced, it is impossible to arrange them into contiguous processing intervals. Therefore, the LP relaxation optimal solution violates the non-preemption requirement of the original problem. This contrasts with the feasible case shown in Figure 6, where processing periods can be arranged into contiguous intervals.

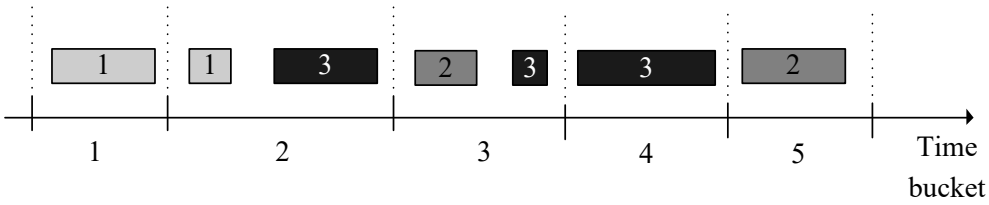


Figure 7. An LP relaxation solution that cannot be rearranged into a feasible non-preemptive schedule.

The feasibility of a solution depends on whether a valid, non-preemptive processing sequence can be constructed. To verify Condition 2, a recursive function employing a depth-first search (DFS) over the enumeration tree is implemented. This function attempts to find a valid permutation of the processing periods for different jobs within each time bucket based on the LP relaxation solution. Two features are incorporated to improve computational efficiency: (1) not all possible sequences need to be explored, as the search terminates immediately once a feasible processing sequence is identified; and (2) if a newly branched node leads to an infeasible partial schedule, no further branching from that node is performed.

Checking Condition 1 requires significantly less computation time than Condition 2; therefore, Condition 1 is verified first. If Condition 1 is not satisfied, the verification of Condition 2 is bypassed. If either condition fails, the optimal solution of the LP-relaxation is deemed infeasible for the original problem. In such cases, the objective value of the LP-relaxation serves as an upper bound for the subtree rooted at the current node.

4.4.3. Fathoming Rules Used in the LP-Relaxation Approach

Let ψ^* denote the incumbent solution during the B&B procedure. Detailed discussions of fathoming rules in B&B for MIP problems are provided in [50]. For a given node under consideration, the following fathoming rules are applied based on the LP-relaxation:

Fathoming rule 1

If the LP-relaxation at a node has no feasible solution, then the node can be fathomed, as no further branching from it will yield a feasible solution.

Fathoming rule 2

Once the LP-relaxation for a node is solved, its objective value serves as an upper bound for the subtree rooted at that node. If this upper bound is no greater than the objective value of the current incumbent solution ψ^*, the node can be fathomed, as it is impossible to find a superior solution within this subtree.

Fathoming rule 3

If the LP relaxation solution at a node satisfies both feasibility Conditions 1 and 2, the node is fathomed. In this case, the solution is recognized as a feasible solution to the original problem, and its objective value is compared with the current incumbent ψ^*. If the new objective value is greater than ψ^*, then ψ^* is updated accordingly.

5. Numerical Experiment

The algorithms and MIP solver were implemented in C using Microsoft Visual Studio 2017 and the Gurobi Optimizer [49]. The C language was selected due to its computational efficiency and flexibility in implementing customized enumeration algorithms. Gurobi was adopted as a state-of-the-art commercial MIP solver due to its strong performance and reliability in solving large-scale optimization problems. All experiments were conducted on a personal computer equipped with an Intel Core i7 CPU (3.40 GHz) and 32 GB of RAM.

5.1. Parameter Settings

This section discusses the parameter settings and control factors used to generate the random problem instances for the experiments.

Known parameters
T: Scheduling horizon, set to 2400 min (based on 5 working days × 8 h/day × 60 min/h).
n: Number of jobs.
α: Ratio of the expected demand to the expected capacity, defined as $\alpha = {n\overline{p}}/{T}$, where $\overline{p}$ is the average processing time of a job.
f: Ratio of the average length of a job time window to the scheduling horizon, defined as $f = {\overline{R}}/{T}$.
v_p: Variation factor of processing time.
v_r: Variation factor of the ratio of a job time window to its processing time.
v_w: Variation factor of weights.
Among these parameters, n, α, f, v_p, v_r, and v_w are considered control factors in the experimental design.
Calculated parameters
$\overline{p}$: Expected processing time of a job, calculated as $\overline{p} = {\alpha T}/{n}$.
$\overline{R}$: Expected length of a job time window, calculated as $\overline{R} = f \times T$.
$\overline{\beta}$: Ratio of the expected length of a job time window to its expected processing time, defined as $\overline{\beta} = {\overline{R}}/{\overline{p}}$.

The parameter $\overline{\beta}$ represents the flexibility or slackness available for scheduling a job. A higher value of $\overline{\beta}$ indicates greater flexibility, allowing more room to schedule jobs within their respective time windows. According to the assumptions stated in Section 3, the processing time of a job must be no greater than the length of its time window. Consequently, this study enforces the condition $\overline{\beta} \geq 1$ when generating random problem instances.

5.2. Problem Generation Procedure

Given known values for parameters T, n, α, f, v_p, v_r, and v_w, the procedure for generating a problem instance is as follows:

1. Compute the expected processing time: $\overline{p} = {\alpha T}/{n}$.
2. Compute the expected slackness ratio: $\overline{\beta} = {fn}/{\alpha}$ (this is derived from $\overline{\beta} = {\overline{R}}/{p}$, $f = {\overline{R}}/{T}$, and $\alpha = {n\overline{p}}/{T}$).
3. For each job $j = 1, 2, \ldots, n$:
   (a) Generate $p_j \sim$ Uniform $[\overline{p}(1 - v_p),\ \overline{p}(1 + v_p)]$.
   (b) Generate $\beta_j \sim$ Uniform $[\overline{\beta}(1 - v_r),\ \overline{\beta}(1 + v_r)]$.
   (c) Compute $R_j = \beta_j \times p_j$.
   (d) Generate $r_j \sim$ Uniform $[0,\ T - fT]$.
   (e) Compute $d_j = r_j + R_j$.
   (f) Generate $w_j \sim$ Uniform $[1 - v_w, 1 + v_w]$.

The input parameters were generated using uniform distributions, which is a common practice in scheduling literature for creating benchmark instances with controlled variability. The use of uniform distributions allows for systematic evaluation of algorithm performance across a wide range of problem settings without introducing bias toward specific parameter patterns. Similar approaches have been adopted in prior scheduling studies to ensure

comparability and generality of computational results. Future research may consider real-world data distributions to further validate the proposed methods.

As is well known, NP-hard problems can lead to rapidly increasing computational time as problem size and combinatorial complexity grow. The parameter settings used to generate random instances can significantly affect solution difficulty. Therefore, after extensive pilot experiments with various parameter configurations, the values of fixed parameters and the levels of control factors were selected to ensure meaningful, well-balanced, and computationally tractable experimental results. The values of the control factors are summarized in Table 1, resulting in 288 combinations ($4 \times 3 \times 3 \times 2 \times 2 \times 2$). For each combination, a random problem instance is generated, yielding 288 instances in total. Each instance is solved using six methods, resulting in 1728 solution runs.

Table 1. Levels of the six control factors.

Control Factors	Levels
n	10, 15, 20, 25
α	1.5, 2.5, 3.5
f	0.4, 0.65, 0.9
v_p	0.4, 0.8
v_r	0.2, 0.4
v_w	0.4, 0.8

5.3. Experiment Setting

Solving LP relaxations requires additional computation time for the B&B method. Therefore, this study investigates whether the use of LP relaxation can effectively reduce the total computation time. Additionally, to evaluate the impact of the proposed ordering framework, the experiments test both IE and B&B methods with and without applying midpoint-based ordering and dominance rules.

Each random instance generated as described in Section 5.2 is solved using the five methods summarized in Table 2. The midpoint-based ordering and dominance rules, introduced in Section 4.3, guide the job selection process. When midpoint-based ordering and dominance rules are not applied, the methods are denoted as "IE_NP" and "B&B_NP" for IE and B&B, respectively, while "IE" and "B&B" refer to the counterparts that utilize these techniques.

Table 2. Abbreviations for the five methods used in our experiments.

Method Abbreviation	Basic Method	Initial Midpoint-Based Ordering	Dominance Rules	Initial Feasible Solution (Heuristic)	LP-Relaxation
IE_NP	IE (C codes)	No	No	No	No
IE	IE (C codes)	Yes	Yes	No	No
B&B_NP	B&B (C codes)	No	No	Yes	Yes
B&B	B&B (C codes)	Yes	Yes	Yes	Yes
MIP	MIP (Gurobi)	-	-	Yes	-

Since the five methods may fail to identify an optimal schedule within a reasonable computational budget, a time limit is imposed based on the number of jobs n, as specified in Table 3. If the computation reaches this predefined time limit without completion, the solution time recorded for that instance is set equal to the time limit.

Table 3. Time limits for different job sizes n.

n	10	15	20	25
Time limit (hours)	1	1	2	4

6. Experiment Results and Analysis

This section presents the results of the computational experiments.

6.1. Effectiveness of Midpoint-Based Ordering → and Dominance Rules

The solution time of NP-hard problems using enumeration-based approaches, including IE, B&B, and the MIP model solved by Gurobi, can vary significantly across problem instances. Even instances generated under identical parameter settings may exhibit substantial differences in computational effort. Such variability is inherent in NP-hard problems, and many instances cannot be solved to optimality within a reasonable time limit. In practical applications, a time limit is typically imposed, and the best feasible solution obtained within that limit is adopted. Therefore, the performance of these methods is inherently time-dependent.

To enable a direct and fair comparison across different problem instances, two normalized metrics are introduced. The normalized objective value of OPT is defined as OPT/Z^*, where Z^* denotes the optimal solution or the best objective value obtained within the time limit across all methods. The normalized computation time of CPU is defined as CPU/t^*, where t^* is the earliest time at which the best-known solution is identified, or the time limit if optimality is not reached. Figures 8–10 present the average normalized performance curves of the tested approaches. The results show that incorporating midpoint-based ordering and dominance rules significantly improves the efficiency of both IE and B&B methods, as evidenced by consistently higher curves compared to their counterparts without these enhancements. Given the negligible computational overhead of these techniques, subsequent analysis focuses on IE and B&B with midpoint-based ordering and dominance rules, along with the MIP model.

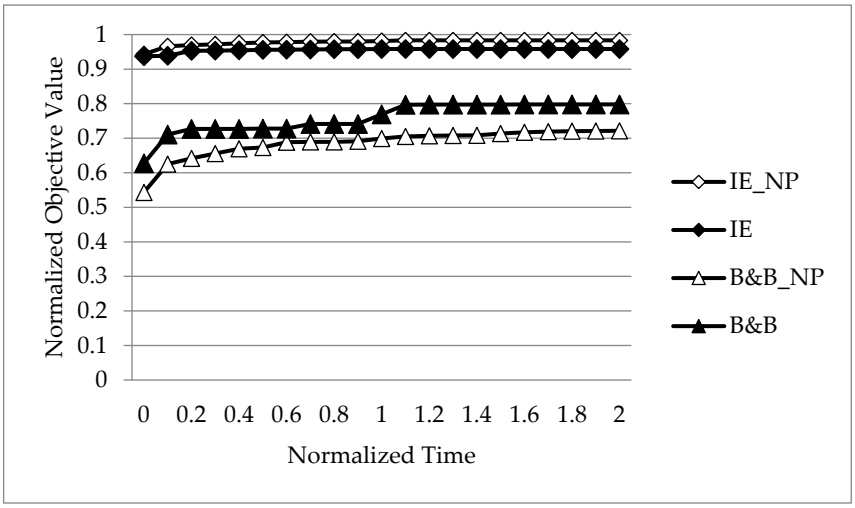


Figure 8. Average normalized objective value versus normalized CPU time for n = 15, illustrating the impact of midpoint-based ordering and dominance rules on the performance of IE and B&B methods.

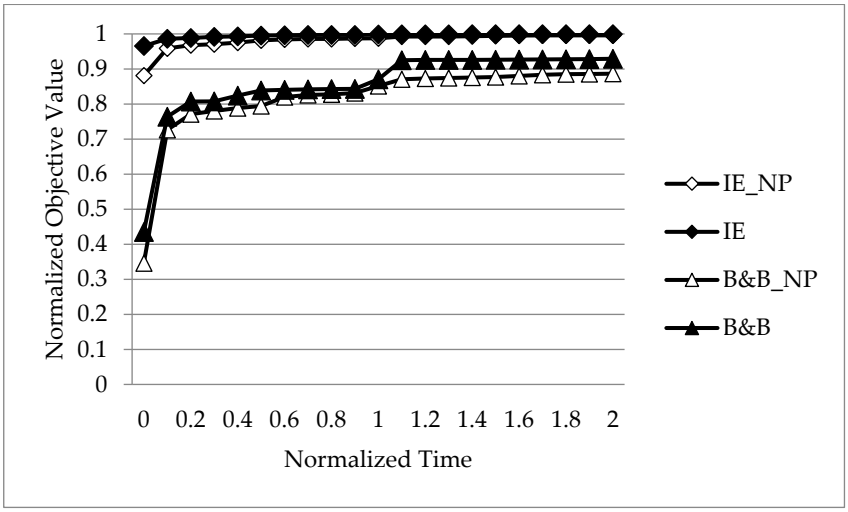


Figure 9. Average normalized objective value versus normalized CPU time for $n = 20$, illustrating the impact of midpoint-based ordering and dominance rules on the performance of IE and B&B methods.

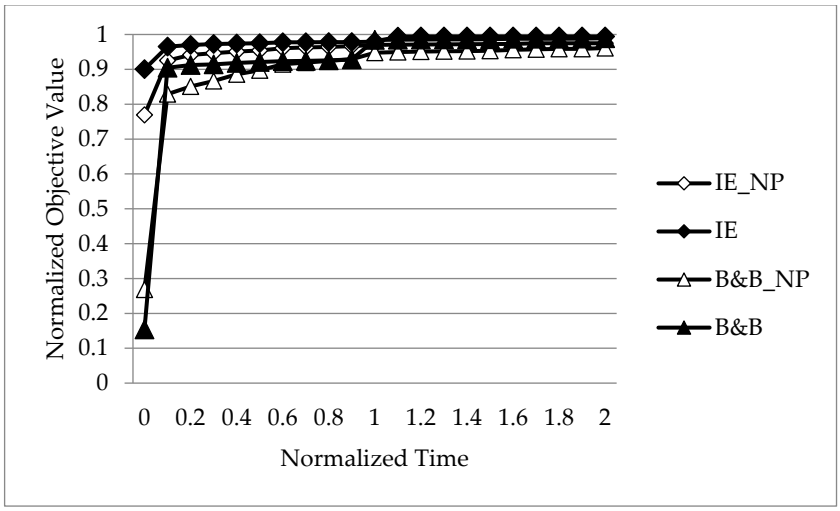


Figure 10. Average normalized objective value versus normalized CPU time for $n = 25$, illustrating the impact of midpoint-based ordering and dominance rules on the performance of IE and B&B methods.

6.2. Effectiveness of Dominance Rules, Fathoming Rules, and LP-Relaxation

As described in Section 4.3, four dominance rules and three fathoming rules were introduced to reduce the search space for both the IE and B&B methods. In both methods, the dominance rules are applied in the order of Rules 1 through 4. For the B&B method, the fathoming rules are subsequently verified in the order of Rules 1, 2, and 3, following the application of the dominance rules. To assess the effectiveness of these rules, the number of times each rule is satisfied during the computation is recorded. For each problem size n, the average number of times each dominance rule is triggered is summarized in Table 4.

The results in Table 4 show that the dominance rules are triggered most frequently in the following order: Rule 2, Rule 1, Rule 4, and Rule 3. Additionally, the impact of these rules on reducing the search space becomes more pronounced as the problem size increases. Notably, the number of rule triggers in the B&B method is lower than that in the IE method for problems of the same size, highlighting the effectiveness of LP-relaxation-based bounding in pruning the search tree. This advantage of the B&B method becomes more significant for larger problem instances.

Table 4. Average number of times each dominance rule is triggered under various levels of n.

Method	n	Dominance Rule			
		1	**2**	**3**	**4**
IE	10	8460.5	14,844.56	20.76	177.53
	15	2.18×10^7	6.52×10^7	631.31	2.65×10^5
	20	2.09×10^{10}	6.43×10^{10}	35,354.28	1.70×10^8
	25	1.89×10^{11}	4.49×10^{11}	3.48×10^6	1.11×10^9
B&B	10	787.15	1068.71	16.29	33.42
	15	2.42×10^5	4.44×10^5	177.06	9933.97
	20	7.98×10^6	1.34×10^7	1603.25	1.75×10^5
	25	4.79×10^7	6.20×10^7	68,685.5	7.00×10^5

The average number of times each fathoming rule is satisfied across various levels of n is presented in Table 5. Fathoming Rule 1 was never triggered because infeasible partial schedules were already eliminated by Dominance Rule 1. Fathoming Rule 2 was satisfied more frequently than Rule 3. When verifying Fathoming Rule 3, Condition 1 is evaluated before Condition 2 because calculating Condition 1 requires minimal computational effort. Although Condition 2 involves a depth-first search (DFS) and consumes more CPU time, it is rarely executed. This is because Condition 1 effectively filters out most infeasible partial schedules in advance. Consequently, Rule 3 is rarely satisfied (averaging less than once per problem) according to the data shown in Table 5.

Table 5. Average trigger counts for various fathoming rules across different levels of n.

n	Fathoming Rule			Conditions of Fathoming Rule 3	
	1	**2**	**3**	**1**	**2**
10	0	173.42	0.14	1.35	0.14
15	0	1.03×10^5	0.21	3.67	0.21
20	0	1.86×10^6	0.40	72.38	0.40
25	0	1.03×10^7	0.42	440.06	0.42

Section 4.4 introduces the LP relaxation bounding scheme to reduce the computation time in the B&B method. To evaluate its effectiveness, Table 6 presents the average number of nodes explored during the search process. The results show that the number of nodes generated in both the IE and B&B methods is significantly lower than that in a complete enumeration tree. Moreover, this reduction becomes more pronounced as the value of n increases, indicating that the LP relaxation is particularly effective for large-scale problems. Despite the substantial node reduction in B&B compared to IE, the IE method still outperforms B&B in terms of total CPU time. This may be attributed to the computational overhead associated with solving the LP-relaxation for the subproblems.

Table 6. Average number of nodes explored for different values of n.

# of Nodes	n			
Method	10	15	20	25
IE	3247.81	7.04×10^6	5.20×10^9	2.60×10^{10}
B&B	442.06	1.68×10^5	3.21×10^6	1.47×10^7
Complete enumerative tree	9.86×10^6	3.55×10^{12}	4.18×10^{18}	4.22×10^{25}

6.3. Performance of Various Methods

The performance of the proposed methods is validated through the experimental results presented in this subsection. For each level of the control factor n (representing the problem size or number of jobs), 72 problem instances were tested. Table 7 tabulates the number of problems that remained unsolved within the predefined time limit for each level of n. As shown, the MIP model (by Gurobi) fails to solve significantly more problems compared to the other two approaches. Notably, when n = 25, 62 out of 72 instances could not be solved within the 4 h time limit.

Table 7. Number of unsolved problem instances for each level of n.

# of Unsolved	***n***			
Method	10	15	20	25
IE	0	0	8/72	26/72
B&B	0	0	4/72	13/72
MIP (Gurobi)	0	28/72	53/72	62/72

Table 8 shows that the computation time for each method scales with the number of jobs. Combining the results from Tables 7 and 8, the IE method outperforms B&B in terms of speed for small-scale problems. However, for large-scale problems, the B&B method solves more instances within shorter computation times than IE. This demonstrates that the bounding scheme provided by the LP relaxation is particularly beneficial to the enumerative approach as the problem size n increases.

Table 8. Average CPU time for each level of n.

Average CPU Time (s)	***n***			
Method	10	15	20	25
IE	0.00	0.98	921.43	5494.88
B&B	0.05	31.09	606.51	3300.81
MIP (Gurobi)	7.30	1582.43	5367.36	12,434.12

The results presented in Tables 7 and 8 indicate that none of the methods can optimally solve large-sized problems within the given time limit. However, as shown in Figures 11–13, the performance curve of the IE method is consistently higher than that of the other two methods, suggesting that the IE approach can obtain high-quality solutions in a shorter computation time. Therefore, the IE method can serve as an effective heuristic approach when a good decision must be made within a limited time, particularly for large-scale problems.

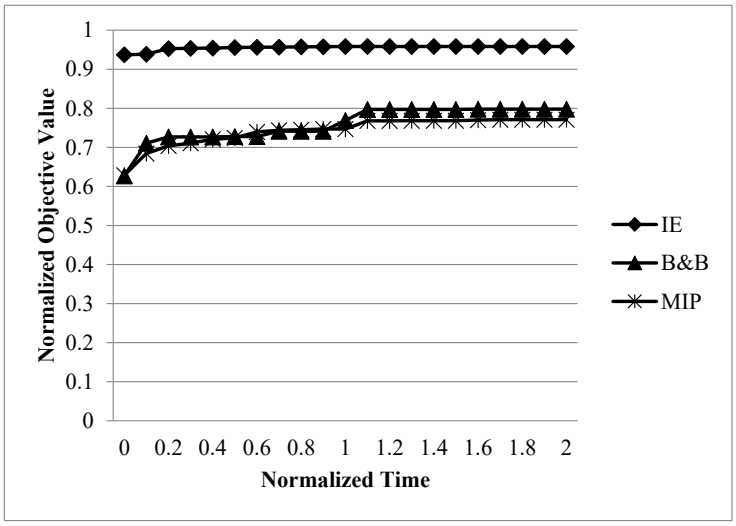


Figure 11. Average normalized objective value versus normalized CPU time for $n = 15$, comparing IE and B&B (with midpoint-based ordering and dominance rules) against the MIP model solved by Gurobi.

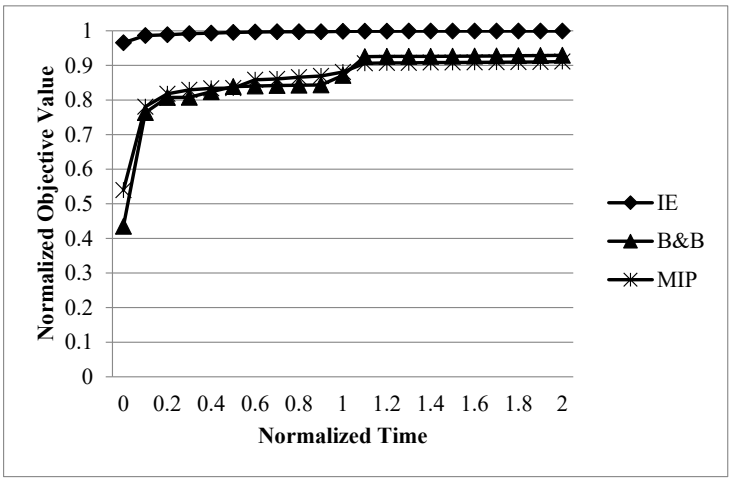


Figure 12. Average normalized objective value versus normalized CPU time for $n = 20$, comparing IE and B&B (with midpoint-based ordering and dominance rules) against the MIP model solved by Gurobi.

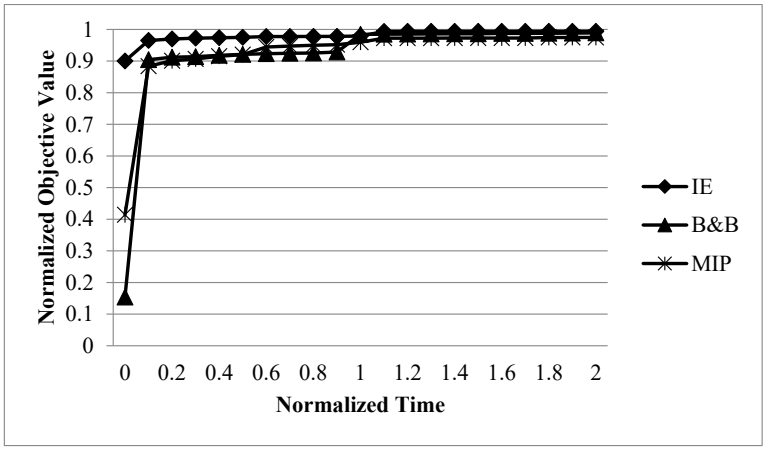


Figure 13. Average normalized objective value versus normalized CPU time for $n = 25$, comparing IE and B&B (with midpoint-based ordering and dominance rules) against the MIP model solved by Gurobi.

7. Conclusions

This study investigates the weighted TMP in single-machine scheduling with time windows, which is a general NP-hard problem. To address this problem, four solution approaches are developed, including a heuristic method, an MIP model solved by Gurobi, an IE method with midpoint-based ordering, dominance rules, and a B&B method enhanced with LP-relaxation.

The computational results demonstrate that the proposed ordering-based framework, which integrates midpoint-based ordering and dominance rules, significantly improves the efficiency of the search process. Among the exact methods, the IE approach performs efficiently for small to medium-sized instances and is capable of producing high-quality solutions in short computation times. For larger problem instances, the B&B method benefits from the LP-relaxation-based bounding scheme, which effectively reduces the search space and improves solution performance. In contrast, the MIP model solved by Gurobi requires substantially longer computation times and is less effective for large-scale instances.

From a practical perspective, the proposed methods can support decision-making in production environments where order selection and scheduling must be performed simultaneously under time window constraints. Examples include make-to-order manufacturing systems and capacity planning for bottleneck machines.

This study has some limitations. The computational experiments are based on randomly generated instances, and real industrial data are not considered. In addition, the model assumes a single-machine environment without sequence-dependent setup times or other operational complexities. Future research may extend this work by incorporating real-world datasets, considering more complex scheduling environments, and developing hybrid approaches that combine exact and heuristic methods.

Author Contributions: Conceptualization, Y.-F.H. and J.-S.L.; methodology, Y.-F.H.; software, J.-S.L.; validation, J.-S.L., C.-H.L. and C.-H.C.; formal analysis, Y.-F.H.; investigation, J.-S.L.; resources, Y.-F.H.; data curation, J.-S.L.; writing—original draft preparation, J.-S.L.; writing—review and editing, Y.-F.H.; visualization, J.-S.L.; supervision, Y.-F.H.; project administration, Y.-F.H. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: The data presented in this study are available on request from the corresponding author due to that (1) the method and parameters that randomly generates the test data are clearly presented in Section 4; (2) The co-author who wrote the programs is my former student and works for private company now, and it will take certain time to generate all the random test data.

Conflicts of Interest: The authors declare no conflict of interest.

Appendix A. Cases for the Relations of Two Jobs' Time Windows

Suppose two jobs $i, j \in J$ and $i \to j$. Let R_i be the range $[r_i, d_i]$ of job i. There are a total of ten possible cases of the relations between the values of r_i, d_i, r_j, and d_j and they are outlined as follows.

Case 1. $r_i < d_i < r_j < d_j$ (R_i does not overlap with R_j)

An example of this case is shown in Figure A1.

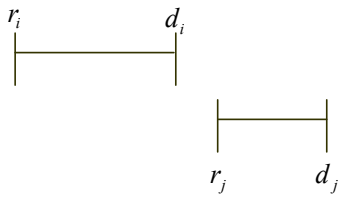


Figure A1. The relation $r_i < d_i < r_j < d_j$.

Case 2. $r_i < d_i = r_j < d_j$ (R_i overlaps with R_j at one time point)

R_i overlaps with R_j only at one time point d_i (=r_j), as shown in Figure A2.

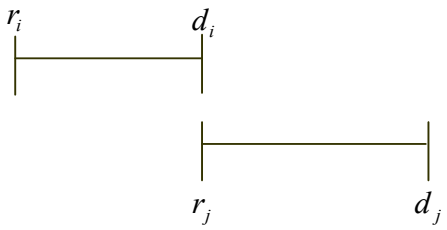


Figure A2. The relation $r_i < d_i = r_j < d_j$.

Case 3. $r_i < r_j < d_i < d_j$ (R_i partially overlaps with R_j)

The interval $[r_i, d_i]$ is shared by R_i and R_j, as shown in Figure A3.

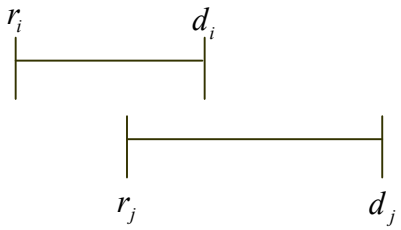


Figure A3. The relation $r_i < r_j < d_i < d_j$.

Case 4. $r_i = r_j < d_i = d_j$ (R_i is identical to R_j)

An example of this case is shown in Figure A4.

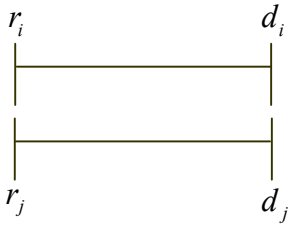


Figure A4. The relation $r_i = r_j < d_i = d_j$.

Case 5. R_i is a subset of R_j and $r_i = r_j < d_i < d_j$

An example of this case is shown in Figure A5.

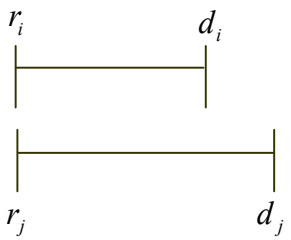


Figure A5. The relation $r_i = r_j < d_i < d_j$.

Case 6. R_j is a subset of R_i and $r_i < r_j < d_i = d_j$

An example of this case is shown in Figure A6.

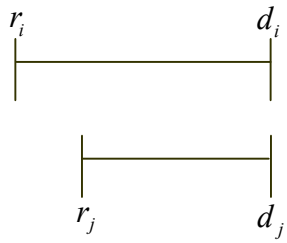


Figure A6. The relation $r_i < r_j < d_i = d_j$.

Case 7. R_j is a subset of R_i, $r_i < r_j < d_j < d_i$, and $d_j - r_i > d_i - r_j$.

An example of this case is shown in Figure A7.

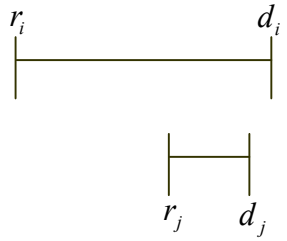


Figure A7. The relation $r_i < r_j < d_j < d_i$.

Case 8. R_j is a subset of R_i, $r_i < r_j < d_j < d_i$, and $d_j - r_i = d_i - r_j$.

An example of this case is shown in Figure A8.

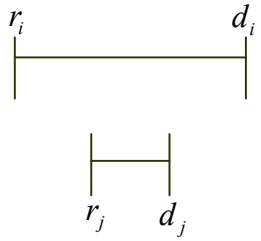


Figure A8. The relation $r_i < r_j < d_j < d_i$, and $d_j - r_i = d_i - r_j$.

Case 9. R_i is a subset of R_j, $r_j < r_i < d_i < d_j$, and $d_j - r_i > d_i - r_j$.

An example of this case is shown in Figure A9.

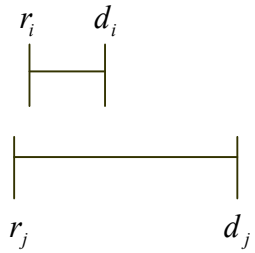


Figure A9. The relation $r_j < r_i < d_i < d_j$, and $d_j - r_i > d_i - r_j$.

Case 10. R_i is a subset of R_j, $r_j < r_i < d_i < d_j$, and $d_j - r_i = d_i - r_j$.

An example of this case is shown in Figure A10.

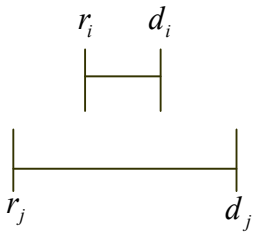


Figure A10. The relation $r_j < r_i < d_i < d_j$, and $d_j - r_i = d_i - r_j$.

Appendix B. Algorithms

This algorithm performs the search in an enumerative tree.

Notation:
H: Solution.
H^*: Current best feasible solution.
J: The sorted set of jobs that are not yet scheduled by midpoint-based ordering.
Main function
{
set H^* = Empty set
set H = Empty set
set J = The sorted set of all jobs by midpoint-based ordering
call recursive function node (J, H)
}
Recursive function node (J, H)
{
if (J is an empty set)
{
if (the objective value of H is better than that of H^*), let $H^* = H$.
}
else
{
for each job j in J, which is sorted by midpoint-based order $\rightarrow$
{
if $H + j$ is dominated by any of the four dominance rules,
then skip current "for" loop.
add job j to the end of H
remove j from J
call recursive function node (J, H)
add j to J
remove j from the last job of H
}
}
}

References

1. Hung, Y.-F.; Huang, C.-C.; Yeh, Y. Real-time capacity requirement planning for make-to-order manufacturing with variable time-window orders. *Comput. Ind. Eng.* **2013**, *64*, 641–652. [CrossRef]
2. Martinelli, R.; Mariano, F.; Martins, C. Single machine scheduling in make to order environments: A systematic review. *Comput. Ind. Eng.* **2022**, *169*, 108190. [CrossRef]
3. Ying, K.; Pourhejazy, P.; Huang, T. Exploring the development trajectory of single-machine production scheduling. *Ann. Oper. Res.* **2024**, 1–26. [CrossRef]

4. Karp, R.M. *Reducibility Among Combinatorial Problems*; Miller, R.E., Thatcher, J.W., Eds.; Complexity of Computer Computations; Springer: Berlin/Heidelberg, Germany, 1972; pp. 85–103.
5. Sahni, S.K. Algorithms for scheduling independent tasks. *J. ACM* **1976**, *23*, 116–127. [CrossRef]
6. Bouška, M.; Šůcha, P.; Novák, A.; Hanzálek, Z. Deep learning-driven scheduling algorithm for a single machine problem minimizing the total tardiness. *arXiv* **2024**, arXiv:2402.14847. [CrossRef]
7. Liu, A.; Yue, D.; Chen, J.; Chen, H. Deep learning for intelligent production scheduling optimization. *Int. J. Simul. Model.* **2024**, *23*, 172–183. [CrossRef]
8. Slotnick, S.A.; Morton, T.E. Selecting jobs for a heavily loaded shop with lateness penalties. *Comput. Oper. Res.* **1996**, *23*, 131–140. [CrossRef]
9. Ghosh, J.B. Job selection in a heavily loaded shop. *Comput. Oper. Res.* **1997**, *24*, 141–145. [CrossRef]
10. Slotnick, S.A.; Morton, T.E. Order acceptance with weighted tardiness. *Comput. Oper. Res.* **2007**, *34*, 3029–3042. [CrossRef]
11. Juraszek, J.; Pesch, E.; Malgorzata, S. Simulated annealing method for maximizing revenue on parallel machines. In Proceedings of the 4th Multidisciplinary International Conference on Scheduling: Theory and Applications, Dublin, Ireland, 10–12 August 2009; pp. 699–701.
12. Arkin, E.M.; Silverberg, E.B. Scheduling jobs with fixed start and end times. *Discret. Appl. Math.* **1987**, *18*, 1–8. [CrossRef]
13. Calinescu, G.; Chakrabarti, A.; Karloff, H.; Rabani, Y. Improved approximation algorithms for resource allocation. In *International Conference on Integer Programming and Combinatorial Optimization*; Springer: Berlin/Heidelberg, Germany, 2002; pp. 401–414.
14. Yang, B.; Geunes, J. A single resource scheduling problem with job-selection flexibility, tardiness costs and controllable processing times. *Comput. Ind. Eng.* **2007**, *53*, 420–432. [CrossRef]
15. Lawler, E.L.; Moore, J.M. A functional equation and its application to resource allocation and sequencing problems. *Manag. Sci.* **1969**, *16*, 77–84. [CrossRef]
16. Lawler, E.L. Sequencing to Minimize Weighted Number of Tardy Jobs. *Rev. Fr. D Autom. Inform. Rech. Oper.* **1976**, *10*, 27–33.
17. Kise, H.; Ibaraki, T.; Mine, H. A solvable case of the one-machine scheduling problem with ready and due times. *Oper. Res.* **1978**, *26*, 121–126. [CrossRef]
18. Lawler, E.L. A dynamic programming algorithm for preemptive scheduling of a single machine to minimize the number of late jobs. *Ann. Oper. Res.* **1990**, *26*, 125–133. [CrossRef]
19. Baptiste, P. Polynomial time algorithms for minimizing the weighted number of late jobs on a single machine with equal processing times. *J. Sched.* **1999**, *2*, 245–252. [CrossRef]
20. Baptiste, P.; Chrobak, M.; Dürr, C.; Jawor, W.; Vakhania, N. Preemptive scheduling of equal-length jobs to maximize weighted throughput. *Oper. Res. Lett.* **2004**, *32*, 258–264. [CrossRef]
21. Nowicki, E.; Zdrzałka, S. A survey of results for sequencing problems with controllable processing times. *Discret. Appl. Math.* **1990**, *26*, 271–287. [CrossRef]
22. Yang, W.H. Scheduling jobs on a single machine to maximize the total revenue of jobs. *Comput. Oper. Res.* **2009**, *36*, 565–583. [CrossRef]
23. Chaurasia, S.N.; Singh, A. Hybrid evolutionary approaches for the single machine order acceptance and scheduling problem. *Appl. Soft Comput.* **2017**, *52*, 725–747. [CrossRef]
24. Silva, Y.L.T.V.; Subramanian, A.; Pessoa, A.A. Exact and heuristic algorithms for order acceptance and scheduling with sequence-dependent setup times. *Comput. Oper. Res.* **2018**, *90*, 142–160. [CrossRef]
25. Li, X.; Ventura, J.A. Exact algorithms for a joint order acceptance and scheduling problem. *Int. J. Prod. Econ.* **2020**, *223*, 107516. [CrossRef]
26. de Weerdt, M.; Baart, R.; He, L. Single-machine scheduling with release times, deadlines, setup times, and rejection. *Eur. J. Oper. Res.* **2021**, *291*, 629–639. [CrossRef]
27. Tarhan, İ.; Oğuz, C. A matheuristic for the generalized order acceptance and scheduling problem. *Eur. J. Oper. Res.* **2022**, *299*, 87–103. [CrossRef]
28. Gerstl, E.; Mosheiov, G. A note: Maximizing the weighted number of Just-in-Time jobs for a given job sequence. *J. Sched.* **2023**, *26*, 403–409. [CrossRef]
29. Kaul, M.; Mnich, M.; Molter, H. Single-Machine Scheduling to Minimize the Number of Tardy Jobs with Release Dates. *arXiv* **2024**, arXiv:2408.12967. [CrossRef]
30. Bar-Noy, A.; Bar-Yehuda, R.; Freund, A.; Naor, J.; Schieber, B. A unified approach to approximating resource allocation and scheduling. *J. ACM* **2001**, *48*, 1069–1090. [CrossRef]
31. Berman, P.; DasGupta, B. Multi-phase algorithms for throughput maximization for real-time scheduling. *J. Comb. Optim.* **2000**, *4*, 307–323. [CrossRef]
32. Chuzhoy, J.; Ostrovsky, R.; Rabani, Y. Approximation algorithms for the job interval selection problem and related scheduling problems. *Math. Oper. Res.* **2006**, *31*, 730–738. [CrossRef]
33. Zhang, L.; Lu, L. Parallel-machine scheduling with release dates and rejection. *4OR* **2016**, *14*, 165–172. [CrossRef]

34. Wang, S.; Ye, B. Exact methods for order acceptance and scheduling on unrelated parallel machines. *Comput. Oper. Res.* **2019**, *104*, 159–173. [CrossRef]
35. Wang, S.; Wu, R.; Chu, F.; Yu, J. An exact decomposition method for unrelated parallel machine scheduling with order acceptance and setup times. *Comput. Ind. Eng.* **2023**, *175*, 108899. [CrossRef]
36. Fanjul-Peyró, L.; Ruiz, R.; Perea, F. Reformulations and an exact algorithm for unrelated parallel machine scheduling problems with setup times. *Comput. Oper. Res.* **2019**, *101*, 173–182. [CrossRef]
37. Mahmoudinazlou, S.; Alizadeh, A.; Noble, J.; Eslamdoust, S. An improved hybrid ICA-SA metaheuristic for order acceptance and scheduling with time windows and sequence-dependent setup times. *Neural Comput. Appl.* **2024**, *36*, 599–617. [CrossRef]
38. Jos, B.C.; Rajendran, C.; Srinivas, S. Order acceptance and scheduling on non-identical parallel machines with dependent setup times: New mixed integer programming formulations. *Flex. Serv. Manuf. J.* **2025**, 1–23. [CrossRef]
39. Baruah, S.; Koren, G.; Mao, D.; Mishra, B.; Raghunathan, A.; Rosier, L.; Shasha, D.; Wang, F. On the competitiveness of on-line real-time task scheduling. *Real-Time Syst.* **1992**, *4*, 125–144. [CrossRef]
40. Koren, G.; Shasha, D. D^{over}: An Optimal On-Line Scheduling Algorithm for Overloaded Uniprocessor Real-Time Systems. *SIAM J. Comput.* **1995**, *24*, 318–339. [CrossRef]
41. Lipton, R.J.; Tomkins, A. Online Interval Scheduling. In *SODA*; Society for Industrial and Applied Mathematics: Philadelphia, PA, USA, 1994; Volume 94, pp. 302–311.
42. Chin, F.Y.; Chrobak, M.; Fung, S.P.; Jawor, W.; Sgall, J.; Tichý, T. Online competitive algorithms for maximizing weighted throughput of unit jobs. *J. Discret. Algorithms* **2006**, *4*, 255–276. [CrossRef]
43. Liu, X.; Xiao, M.; Li, W.; Zhu, Y.; Ma, L. Algorithms for single machine scheduling problem with release dates and submodular penalties. *J. Comb. Optim.* **2023**, *45*, 105. [CrossRef]
44. Bar-Noy, A.; Guha, S.; Naor, J.; Schieber, B. Approximating the throughput of multiple machines in real-time scheduling. *SIAM J. Comput.* **2001**, *31*, 331–352. [CrossRef]
45. Bar-Noy, A.; Guha, S.; Katz, Y.; Naor, J.S.; Schieber, B.; Shachnai, H. Throughput maximization of real-time scheduling with batching. *ACM Trans. Algorithms* **2009**, *5*, 18. [CrossRef]
46. Og, C.; Salman, F.S.; Yalçın, Z.B. Order acceptance and scheduling decisions in make-to-order systems. *Int. J. Prod. Econ.* **2010**, *125*, 200–211.
47. Graham, R.L.; Lawler, E.L.; Lenstra, J.K.; Rinnooy Kan, A.H.G. Optimization and approximation in deterministic sequencing and scheduling: A survey. *Ann. Discret. Math.* **1979**, *3*, 287–326.
48. Mathews, G.B. On the partition of numbers. *Proc. Lond. Math. Soc.* **1897**, *28*, 486–490. [CrossRef]
49. Gurobi. 2025. Available online: https://www.gurobi.com/ (accessed on 17 March 2026).
50. Hillier, F.S.; Lieberman, G.J. *Introduction to Operations Research*, 11th ed.; McGraw-Hill Education: Columbus, OH, USA, 2021.

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.