Extended Arc-Time-Indexed Formulation and Exact Approach for Flowshop Scheduling

Long Chen*[1], Lan Lu†[1], and Lindong Liu‡[1]

[1]School of Management, University of Science and Technology of China, Hefei 230026, China

November 04, 2025

Abstract

This study addresses the classical flow shop scheduling problem, involving a finite set of jobs and machines, where each job must be processed on all machines in a fixed order. The objective is to determine a non-preemptive schedule that minimizes the makespan, defined as the maximum job completion time. This problem arises frequently in industrial settings, such as photolithography in semiconductor manufacturing and production lines in the automotive industry. However, it becomes strongly NP-hard when the number of machines is three or more, and additional practical constraints further exacerbate its complexity. Traditional approaches typically adopt the Position-Based Formulation (PBF), which is effective only for small to medium-sized instances. This paper introduces the Arc-Time-Indexed Formulation (ATIF) for flow shop scheduling for the first time, leveraging a network flow representation. By modifying the network structure (such as incorporating virtual initial arcs and idle arcs), the model is adapted to capture the sequential nature of flow shop operations. A branch-and-price algorithm is then developed to solve medium-scale instances exactly. Extensive computational experiments demonstrate that ATIF significantly outperforms the PBF in both modeling flexibility and solution efficiency.

Keywords: Flow shop scheduling, Arc-time-indexed formulation, Dantzig-Wolfe decomposition, Branch-and-Price

1 Introduction

This research investigates a canonical flow shop scheduling problem, characterized by a finite set of jobs $J = \{1, ..., n\}$ and a finite set of machines $M = \{1, ..., m\}$. Each job $j \in J$ is associated with a prescribed processing time $p_{jk} \in \mathbb{N}\backslash\{0\}$ on each machine $k \in M$. The objective is to schedule these jobs in a sequential manner across all machines without allowing pre-emption, thereby minimizing the makespan, defined as the maximum completion time of all jobs. The aforementioned description can be represented by the three-field notation for scheduling problems as $F_m||C_{\max}$ (Pinedo, 2012).

The flow shop scheduling problem has extensive applications in real-world scenarios. For instance, in the photolithography area of semiconductor manufacturing, to replicate the patterns from the photomask onto the silicon wafer, the wafer must sequentially undergo processes such as photoresist coating,

*Email: lchen1307@ustc.edu

†Corresponding author. Email: lluae@ustc.edu.cn

‡Email: ldliu@ustc.edu.cn

exposure, and development (Uzsoy et al., 1992; Gupta and Sivakumar, 2006; Sha et al., 2006; Park and Morrison, 2014; Yugma et al., 2015; Knopp et al., 2017; Madathil et al., 2018; Zhang et al., 2018; Ghasemi et al., 2020; Huang et al., 2025). Similarly, in the automotive manufacturing process, vehicles must go through a series of operations including stamping, welding, painting, and final assembly. Accurately and efficiently solving the flow shop scheduling problem is highly beneficial for guiding actual production in factories (Htay et al., 2013; Kshatra et al., 2019; Fangrit et al., 2020; Muñoz-Sánchez et al., 2024; Li et al., 2025; Yan and Wang, 2025). Unfortunately, the exact solution of the flow shop scheduling problem remains a challenging issue. Although the two-machine flow shop scheduling problem can be optimally solved using Johnson's algorithm (Johnson, 1954; Campbell et al., 1970), when the number of machines is greater than or equal to three, the problem can be proven to be strongly NP-hard by reduction from the 3-Partition problem (Pinedo, 2012, Theorem 6.1.7). Once additional real-world constraints, such as limited waiting times and sequence-dependent setup times, are considered, the complexity of solving this problem increases exponentially (Yang and Chern, 1995; An et al., 2016; Kim and Lee, 2019).

Previous studies usually employ the Position-Based Formulation (PBF) for modeling (Yang and Chern, 1995; Su, 2003; An et al., 2016; Kim and Lee, 2019; Gmys, 2022; Shabtay and Gerstl, 2024). As the name implies, this type of model requires binary decision variables x_{ir} to indicate whether job i is scheduled to be processed at position r. In addition, the position-based formulation necessitates integer decision variables $T_{[r]k}$ to represent the start times of processing at each position on the machines. It is worth noting that exact algorithms based on position-based formulations are only capable of solving instances of small to medium-sized flow shop scheduling problems within a reasonable timeframe. This is clearly at odds with the industrial pursuit of highly efficient solutions and guidance. This discrepancy prompts us to contemplate whether there exists a superior modeling approach that could more effectively solve the flow shop scheduling problem exactly.

We observe that the flow shop scheduling problem possesses the characteristics of a composite flow. Specifically, from the perspective of the jobs, the sequential completion of all operations from the first to the last machine exhibits a typical flow pattern. From the perspective of the machines, all machines are required to process the jobs in the same order, thereby also demonstrating the same flow characteristic. Consequently, our research opts to model the scheduling problem using a network flow diagram, with the aim of proposing a superior modeling approach to efficiently solve the flow shop scheduling problem exactly.

Fortunately, despite the absence of existing literature on modeling and solving the flow shop scheduling problem using network flow diagrams, we have identified an Arc-Time-Indexed Formulation (ATIF) based on network flow diagrams (Pessoa et al., 2010; Oliveira and Pessoa, 2020). This modeling approach has been extensively applied to parallel machine scheduling problems, achieving excellent results and enabling the exact solution of medium to large-scale parallel machine scheduling problems. Moreover, by redefining the nodes and arcs in the network diagram, constraints such as limited waiting times and sequence-dependent setup times can be seamlessly incorporated into the model without altering the mathematical formulation. This process is cost-effective and does not compromise solution efficiency. Therefore, our work focuses on extending the ATIF to the flow shop scheduling problem through the rational design and adjustment of the network diagram structure, and developing suitable algorithms to solve this problem effectively. The main contributions of our research are listed as follows.

- We are the first to introduce the Arc-Time-Indexed formulation (ATIF) into the solution of the

flow shop scheduling problem. By modifying the structure of the network diagram, such as incorporating idle arcs and initial arcs, we adapt it to the characteristics of flow shop scheduling.

- We have designed a branch-and-price (B&P) algorithm to efficiently solve the flow shop scheduling problem based on the Arc-Time-Indexed formulation (ATIF). This algorithm is capable of obtaining exact solutions for instances of medium scale.
- The models and algorithms proposed in this study were applied to the generated test instances. Through rigorous and comprehensive numerical experiments, the superiority of the Arc-Time-Indexed formulation (ATIF) over the Position-Based formulation (PBF) was demonstrated.

The remainder of this paper is structured as follows. In §2, we provide a comprehensive review of the existing literature. In §3, we introduce a network-based representation to characterize the classical flow-shop scheduling problem and formulate the Arc-Time-Indexed Formulation (ATF) accordingly. §4 presents the design of a branch-and-price algorithm to solve the ATF model. In §5, a branch-and-bound algorithm is introduced as a benchmark to address the Position-Based Formulation (PBF). Extensive numerical experiments are conducted in §6 to validate the proposed approach. Finally, §7 concludes the paper and outlines directions for future research.

2 Previous Work

This section is structured into four parts. First, we introduce the Time-Indexed Formulation (TIF) and review its development and computational challenges. Second, we discuss advancements in alternative formulations such as the Bucket-Indexed formulations (BIF) and Arc-Time-Indexed Formulations (ATIF). Third, we summarize recent efforts to apply graph-based scheduling models in various application domains. Lastly, we present studies on Position-Based Formulation (PBF) for flow shop scheduling and highlight its limitations when applied to complex real-world settings.

2.1 Time-Indexed Formulations for Scheduling Problems

The emergence of the Time-Indexed Formulation (TIF) marked the initial application of graph theory in modeling scheduling problems. Compared to formulations for other single-machine or parallel-machine scheduling problems, TIF provides tighter bounds through its linear relaxation, thereby better guiding the design of exact and approximation algorithms. Unfortunately, TIF involves a large number of variables and constraints, making its linear relaxation difficult to solve and computationally intensive. Additionally, this model is highly dependent on time discretization.

To address these issues above, scholars have leveraged the unique mathematical properties of the model to eliminate unnecessary variables and constraints. From this perspective, appropriate modeling approaches are often required, supplemented by column generation and cut generation to incorporate effective variables and constraints. Van Den Akker et al. (1999) presents a polyhedral approach to single-machine scheduling problems, providing a complete characterization of facet-inducing inequalities for TIF and developing a branch-and-cut algorithm to solve the problem by incorporating these inequalities. Van Den Akker et al. (2000) explores the application of Dantzig-Wolfe decomposition and column generation techniques to address the computational challenges with TIF, demonstrating that these methods can effectively reduce the problem size and improve the efficiency of solving TIF while still allowing for the use of cut generation techniques. Avella et al. (2005) propose a Lagrangian heuristic for large-scale single-machine scheduling problems, leveraging the TIF formulation to achieve near-optimal solutions with small duality gaps in reasonable time. Pan and Shi (2007) demonstrates

the equivalence of the max-min transportation lower bound and the time-indexed lower bound for the single-machine scheduling problems. Bigras et al. (2008) presents a column-generation-based solution approach for the total weighted tardiness problem using TIF, introducing a time-decomposition method to accelerate the computation of strong bounds and embedding it into a branch-and-bound algorithm to solve large-scale instances optimally. Sourd (2009) introduces a reinforced Lagrangian relaxation of TIF to obtain a stronger lower bound and a new branching scheme to build blocks by merging adjacent tasks, significantly improving the efficiency of solving both general and common due date cases.

2.2 Advances in Alternative Formulations of TIF

Scholars also have accelerated solution efficiency by reconstructing the network graph and the form of discretized time. Boland et al. (2016) uses the bucket-indexed-formulation (BIF) for nonpreemptive single machine scheduling problems to partition the planning horizon into equal-length periods, or buckets, significantly reducing the number of variables. Furthermore, BIF also retains the advantage of TIF, such as its parsiony and versatility, while mitigating the issues of large model size. Kowalczyk and Leus (2018) creatively introduce the zero-suppressed binary decision diagrams (ZDDs) to efficiently solve the pricing problem in the branch-and-price framework for parallel machine scheduling problem, thereby enhancing computational performance. Kowalczyk et al. (2024) present a flow-based formulation using decision diagrams for parallel machine scheduling with weighted tardiness, which provides stronger bounds than the time-indexed formulation and demonstrates competitive performance in solving large instances.

The Arc-Time-Indexed Formulation (ATIF) can be regarded as an extension of the Time-Indexed Formulation (TIF) model. Research has shown that ATIF dominates TIF because the former's linear relaxation is tighter than that of the latter. Pessoa et al. (2010) introduces the ATIF for the identical parallel machine scheduling problem, where each variable is indexed by a pair of jobs and a completion time. The authors develop a branch-cut-and-price algorithm, which for the first time, enables the optimal solution of medium-sized instances (up to 100 jobs) of this problem. Oliveira and Pessoa (2020) improves upon the previous research by introducing new overload elimination cuts and a genetic separation algorithm, enhancing the branch-cut-and-price algorithm for the parallel machine scheduling problems. They also switch to a TIF for the MIP solver, enhancing it with projected cuts. The improved algorithm, BCP-PMWT-OTI, can solve instances with up to 100 jobs and 4 machines, and increases solving efficiency by 95.7%. Morais et al. (2024) proposes exact and heuristic algorithms for the single-machine scheduling problem with sequence-dependent setup times and release dates. The exact method uses a branch-and-price algorithm with an ATIF, while the heuristic approach employs an iterated local search framework. Tested on 1800 instances, the algorithms outperform existing methods, with the B&P algorithm solving 42.7% optimally and the ILS-BS heuristic providing competitive results.

The above research indicates that scholars have gained a deep understanding of the polyhedral properties of ATIF for general single-machine and parallel-machine scheduling problems, and are able to accelerate the solution process using highly effective cutting planes and algorithmic techniques. However, the current research on the ATIF model focuses solely on single-machine and parallel-machine problems. To our knowledge, there is hardly any research that extends it to other machine environments.

2.3 Graph-Based and Network Flow Models

Graph theory, particularly network flow models, has demonstrated significant advantages in many other combinatorial optimization and operations management problems. Avella et al. (2017) address the runway scheduling problem, developing a time-indexed MIP approach that can solve real-life instances

efficiently within the required real-time constraints. They introduce new valid inequalities and fixing procedures to enhance the formulation. Their results show significant improvements in computational efficiency and solution quality due to the sophisticated formulation. Cire et al. (2019) developed a network-based formulation (NBF) for scheduling clinical rotations in medical education. The NBF employs decision diagrams to model feasible schedules, offering significant improvements in computational efficiency and cost reduction compared to traditional MILP models. This study, based on data from the American University of the Caribbean, shows that the NBF approach can achieve substantial cost savings and provide valuable insights for managerial decision-making. Martin-Iradi et al. (2022) propose a novel solution method for the multiport berth allocation problem (MBAP) using a generalized set partitioning problem (GSPP) formulation. Their approach, which leverages column generation and branch-and-cut techniques, highlights the power of graph-based formulations in optimizing large-scale logistics and transportation problems. Liu et al. (2024) presents a study on drone resupply with multiple trucks and drones for on-time delivery along given truck routes. This research addresses the challenge of delivering late-available packages in urban areas by proposing a drone resupply model that integrates multiple trucks and drones. The authors develop a time-expanded network flow model with side constraints and a greedy algorithm to solve the problem efficiently.

2.4 Position-Based Formulations for Flow Shop Scheduling

A substantial body of literature has attempted to solve flow shop scheduling problems based on Position-Based Formulation (PBF) with various constraints using various algorithms, particularly the branch-and-bound framework, to solve these problems as precisely as possible. These constraints, originating from real industrial production environments, include limited waiting time constraints, sequence-dependent setup times, and re-entrant production, among others. Yang and Chern (1995) address a two-machine flow shop scheduling problem with limited waiting time, aiming to minimize makespan. They propose a branch-and-bound algorithm that solves instances with up to 80 jobs efficiently, especially when waiting times are generous. Su (2003) study a hybrid two-stage flow shop scheduling problem with limited waiting time constraints. They propose a mixed integer programming model and a two-phase heuristic. The heuristic achieves over 96% solution quality on average with short runtimes and can efficiently solve instances with up to 20 jobs. Kim and Lee (2019) examine a three-machine flow shop scheduling problem with overlapping waiting time constraints and demonstrate that their proposed algorithm can efficiently solve the instances with up to 50 jobs. An et al. (2016) address a two-machine flowshop scheduling problem with limited waiting time constraints and sequence-dependent setup times, proposing a branch-and-bound algorithm that can solve instances with up to 30 jobs efficiently. Shabtay and Gerstl (2024) study a two-machine flow shop scheduling problem with rejection options, aiming to minimize the sum of makespan and total rejection costs. This research proves the problem is NP-hard, provides a mixed integer programming formulation, and develops approximation algorithms and FPTAS, while also exploring its parameterized complexity. These researches above effectively address flow shop scheduling problems with 2 to 3 machines and a medium number of jobs.

However, extending the number of machines or solving larger instances with more jobs presents significant challenges, which inspires us to develop more sophisticated formulations to address this issue.

Furthermore, Gmys (2022) propose a parallel branch-and-bound algorithm using GPU-accelerated supercomputers to solve large-scale permutation flow shop scheduling problems. Their method successfully solves instances with up to 200 jobs and 20 machines, showing that exact methods can scale to very large problems with the help of high-performance computing.

However, this approach relies heavily on access to large-scale computing resources, which limits its practical applicability in typical industrial settings.

3 Graph Definition and Formulation

In this section, we provide a formal definition of the graph-based modeling for the flow shop scheduling. Subsequently, we introduce two types of mathematical formulations, namely the Arc-Time-Indexed Formulation (ATIF) and the Position-Based Formulation (PBF). Through theoretical analysis, we demonstrate the superiority of ATIF over PBF.

3.1 Graph Definition

We construct the Arc–Time–Indexed Formulation (ATIF) for the flow shop scheduling problem based on the predefined network representation of each machine. Let $J = \{1, \ldots, n\}$ denote the set of real jobs, and let $M = \{1, \ldots, m\}$ denote the set of machines arranged in the same technological order for all jobs. A dummy job 0 is introduced to represent the start and end of each machine's schedule, and we define $J_0 = J \cup \{0\}$. The time horizon is discretized as $\mathcal{T} = \{0, 1, \ldots, T\}$, where T represents an upper bound on the makespan in the optimal schedule.

Let the graph be denoted by $G = (V, A)$, where $V = \{(i, t) : i \in J_0, t \in \{0, 1, 2, ..., T\}\}$ and $A = \{((i, t - p_i), (j, t)) : (i, t - p_i) \in V, (j, t) \in V\}$. Briefly, the arcs in the set A can be abbreviated as $(i, j)^t$. Each node (i, t) signifies the state where job i is completed at time t. Finally, a path from node $(0, 0)$ to node $(0, T)$ represents a feasible schedule for a single machine. Since the research problem is the flow shop scheduling with m machines, a feasible solution in the network graph should consist of m complete and interdependent paths.

Furthermore, we need to make two adjustments to the network graph structure to adapt it to our research problem while ensuring its feasibility. First, add virtual initial arcs. In the original network graph, the source node $(0, 0)$ is only connected to the node (i, p_{i1}) representing the first job i processed on machine 1. This graph structure violates the basic flow conservation constraint, so we need to link $(0, 0)$ to the $|M| - 1$ nodes $(i, \sum_{k \in \mathcal{M}} p_{ik})$ representing the first job i processed on machines other than the first one. Second, add idle arcs. In the scheduling represented by the original network graph, there may be violations of the basic logical constraints of flow shop scheduling, that is , a job must be processed on the $(k-1)$th machine before it can be processed on the kth machine. To avoid violations of the basic constraint, we need to link node (i, t) with node $(i, t + 1)$ to ensure that job i can only be processed on the kth machine after it has been processed on the $(k-1)$th machine. To more precisely illustrate the adjustments we have made to the network graph, we will use Example 1 for clarification.

Example 1. Consider a flow shop scheduling problem with five jobs (denoted as J_1, J_2, J_3, J_4, J_5) to be processed on two machines (denoted as M_1 and M_2). Each job must be processed sequentially on both machines without preemption. The processing times for each job on each machine are given in Table 1.

Table 1: Processing Times for a Example of Flow Shop Scheduling

Machine k	Job J_1	Job J_2	Job J_3	Job J_4	Job J_5
M_1	18	10	17	12	16
M_2	14	19	15	14	16

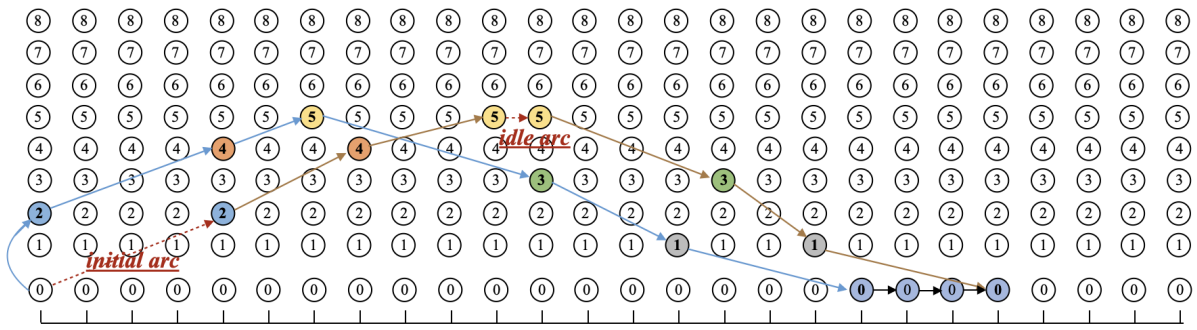

Figure 1: Graph Representation of Feasible Solution of Example 1.
Note: This figure illustrates only the sequencing of feasible solutions, not the exact numerical values.

Remark 1. Figure 1 represents a feasible solution, for an instance with job sequence of job 2 $\rightarrow$ job 4 $\rightarrow$ job 5 $\rightarrow$ job 3 $\rightarrow$ job 1. And in this figure, the decision variable $x_{02}^{0} = x_{24}^{10} = x_{45}^{22} = x_{53}^{38} = x_{31}^{55} = x_{10}^{73} = 1$, which represents the feasible schedule in machine 1. Similarly, the scheduling decision variable for machine 2 is $x_{02}^{10} = x_{24}^{29} = x_{45}^{43} = x_{53}^{59} = x_{31}^{74} = x_{10}^{88} = 1$. It is worth mentioning that the above network diagram can express the same meaning as the Gantt chart shown in Figure 2.

3.2 Arc-Time-Index Formulation

Each feasible solution can be represented as a flow through a time–expanded network where nodes correspond to jobs at specific time points and arcs represent feasible transitions between jobs in time. Let x_{ijt}^{k} be a binary decision variable that equals 1 if arc $(i,j)^t \in \mathcal{A}$ on machine k is used in the solution, that is, if job i finishes and job j starts processing on machine k at time t; otherwise, it is 0. This variable structure simultaneously captures both sequencing (the order of jobs) and timing (the start time) information, allowing the model to represent idle times naturally through dummy arcs.

The processing time of job j on machine k is denoted by p_{jk}, with $p_{0k} = 0$ for the dummy job. The total processing horizon T is selected large enough to cover all feasible completion times, typically $T \geq \sum_{j \in J} \sum_{k \in M} p_{jk}$.

To enforce a common processing order across all machines, we introduce additional binary sequencing variables y_{ij}, where $y_{ij} = 1$ if and only if job i immediately precedes job j in the global permutation sequence, and $y_{ij} = 0$ otherwise. These variables link the job transitions across machines and ensure the permutation flow shop structure.

Furthermore, the continuous variable α represents the makespan, i.e., the completion time of the last job on the last machine, which is minimized in the objective function.

Finally, the complete ATIF for flow shop scheduling is as follows:

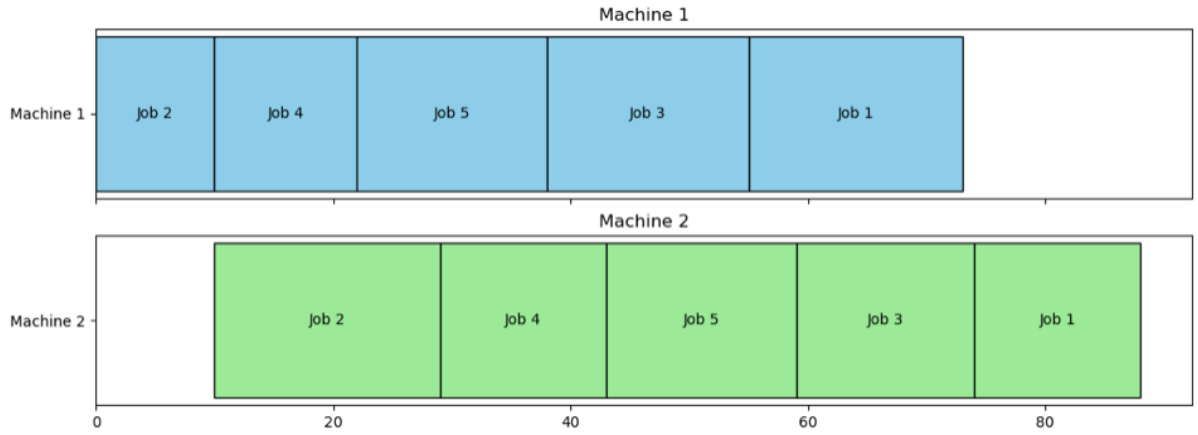

Figure 2: Gantt Chart of Feasible Solution of Example 1

$$
\begin{aligned}
\text{(ATIF)} \quad & \min_{\alpha,\, x,\, y} \quad \alpha && && \text{(1a)}\\
& \text{s.t.} \quad \sum_{j\in J_0} x^k_{0j0} = 1, && \forall k \in M, && \text{(1b)}\\
& \sum_{i\in J_0\setminus\{j\}} \sum_{t=p_{ik}}^{T-p_{jk}} x^k_{ijt} = 1, && \forall j \in J,\ \forall k \in M, && \text{(1c)}\\
& \sum_{\substack{j\in J_0\setminus\{i\},\\ t-p_{jk}\geq 0}} x^k_{jit} - \sum_{\substack{j\in J_0\setminus\{i\},\\ t+p_{ik}+p_{jk}\leq T}} x^k_{ij,t+p_{ik}} = 0, && \forall i \in J,\ t = 0,\dots,T-p_{ik},\ \forall k \in M, && \text{(1d)}\\
& \sum_{\substack{j\in J_0,\\ t-p_{jk}\geq 0}} x^k_{j0t} - \sum_{\substack{j\in J_0,\\ t+p_{jk}+1\leq T}} x^k_{0j,t+1} = 0, && t = 0,\dots,T-1,\ \forall k \in M, && \text{(1e)}\\
& \sum_{t\leq T} x^k_{ijt} = y_{ij}, && \forall (i,j)^t \in \mathcal{A},\ \forall k \in M, && \text{(1f)}\\
& \sum_{j\in J_0\setminus\{i\}} y_{ij} = 1, && \forall i \in J_0, && \text{(1g)}\\
& \sum_{j\in J_0\setminus\{j\}} y_{ij} = 1, && \forall j \in J_0, && \text{(1h)}\\
& \alpha \geq \sum_{i\in J_0} \sum_{t\leq T} (t + p_{jm}) x^m_{ijt}, && \forall j \in J, && \text{(1i)}\\
& x^k_{ijt}, \in \{0,1\}, \forall i \in J_0, && \forall j \in J_0\setminus\{i\},\ \forall k \in M,\ t = p_{ik}, ..., T-p_{jk}, && \text{(1j)}\\
& y_{ij} \in \{0,1\}, && \forall i \in J_0,\ \forall j \in J_0\setminus\{i\}, && \text{(1k)}\\
& \alpha \geq 0. && && \text{(1l)}
\end{aligned}
$$

The objective function (1a) is defined as the maximum completion time of all jobs on the final machine. Capacity constraints (1b) ensure that each machine k starts its schedule with a single outgoing arc from the dummy job 0, representing the beginning of processing on that machine. Assignment constraints (1c) guarantee that each real job j is processed exactly once on every machine k. Flow conservation constraints (1d) preserve the continuity of processing on each machine, ensuring that if a job finishes at time t, the next job on the same machine can only start afterward. Idleness constraints (1e) allow the flow to remain at the dummy job, thus permitting idle periods between consecutive jobs on each machine. Linking constraints (1f) connect the machine–specific arc–time variables x^k_{ijt} with the global sequencing variables y_{ij}, enforcing a consistent job order across all machines. Outgoing–flow constraints (1g) ensure that each job (including the dummy job) has exactly one immediate successor in the global sequence. Incoming–flow constraints (1h) ensure that each job (including the dummy job) has exactly one immediate predecessor in the global sequence. Makespan constraints (1i) define α as an upper bound on the completion time of every job j on every machine k; it can equivalently be restricted to the last machine. Binary constraints (1j) and (1k) define the decision variables x^k_{ijt} and y_{ij} as binary, reflecting discrete sequencing and scheduling decisions. Nonnegativity constraint (1k) enforces that the makespan variable α must be nonnegative.

3.3 Generalized Set Partitioning Formulation

In the Arc-Time-Indexed Formulation (ATIF) introduced in the previous section, each machine $k \in M$ is associated with a set of arc-time variables x^k_{ijt} that form a feasible flow over time. Constraints (1d) and (1e) ensure that the processing of jobs follows a time-feasible sequence of arcs from the dummy job 0 to itself through possible idle periods on each machine. This network structure can be represented as a directed acyclic graph (DAG) $G^k = (V^k, A^k)$, where each node $(i,t) \in V^k$ corresponds to the completion of job i at time j, and each arc $(i,j)^t \in A^k$ denotes the event that job i finishes and job j starts processing at time t on machine k. The graph implicitly captures both processing and idle transitions, since arcs of the form $(0,0) \to (0,t)$ represent idle time before any job starts.

Definition 1. (Pseudo-schedule (Van Den Akker et al., 2000)) A pseudo-schedule (also called a path) on machine k is defined as any feasible path from the source node $(0,0)$ to the sink node $(0,T)$ in G^k. Each pseudo-schedule therefore corresponds to a complete sequence of operations and idle intervals that satisfies the machine interval flow conservation and capacity constraints.

Because these paths automatically respect the machine-level temporal feasibility encoded in constraints (1d) and (1e), they provide a naturall building block for the Dantzig-Wolfe decomposition of the ATIF (Dantzig and Wolfe, 1960). In other words, the feasible region of the original formulation can be described as a convex combination of such pseudo-schedules for each machine.

Let $\mathcal{P}_k$ denote the set of all pseudo-schedules on machine k. For each path $p \in \mathcal{P}_k$, we define a binary constant q^{pk}_{ijt} that euals 1 if and only if arc $(i,j)^t$ is included in path p, and 0 otherwise. A non-negative continuous variable λ^k_p is then introduced to indicate whether pseudo-schedule p is selected in the overall solution. Each variable λ^k_p thus represents a column in the Dantzig-Wolfe master problem, corresponding to one feasible schedule for machine k.

The relationship between the original arc-time variables and the pseudo-schedule columns is expressed as:

$$x_{ijt}^k = \sum_{p \in \mathcal{P}_k} q_{ijt}^{pk} \lambda_p^k, \qquad \forall (i,j)^t \in \mathcal{A}^k,\ \forall k \in M. \tag{2}$$

By substituting the polyhedral representation in (2) into the ATIF model and replacing the variable x_{ijt}^k, we can obtain the corresponding generalized set partitioning (GSP) formulation:

$$\begin{aligned}
\text{(GSP)} \quad \min_{\alpha,\lambda,y} \quad & \alpha && && \text{(3a)}\\
\text{s.t.} \quad & \sum_{p \in \mathcal{P}_k} \sum_{(0,j)^0 \in A} q_{0j0}^{pk} \lambda_p^k = 1, && \forall k \in M, && \text{(3b)}\\
& \sum_{p \in \mathcal{P}_k} \sum_{i \in J_0 \setminus \{j\}} \sum_{t \le T} q_{ijt}^{pk} \lambda_p^k = 1, && \forall j \in J, \forall k \in M, && \text{(3c)}\\
& \sum_{p \in \mathcal{P}_k} \sum_{t \le T} q_{ijt}^{pk} \lambda_p^k = y_{ij}, && \forall (i,j)^t \in \mathcal{A}, \forall k \in M, && \text{(3d)}\\
& \sum_{j \in J_0 \setminus \{i\}} y_{ij} = 1, && \forall i \in J_0, && \text{(3e)}\\
& \sum_{i \in J_0 \setminus \{j\}} y_{ij} = 1 && \forall j \in J_0, && \text{(3f)}\\
& \alpha \ge \sum_{p \in \mathcal{P}_m} \sum_{i \in J_0} \sum_{t \le T} (t + p_{jm}) q_{ijt}^{pm} \lambda_p^m, && \forall j \in J, && \text{(3g)}\\
& \lambda_p^k \ge 0, && \forall p \in \mathcal{P}_k, \forall k \in M, && \text{(3h)}\\
& 0 \le y_{ij} \le 1, && \forall i \in J_0,\ \forall j \in J_0 \setminus \{i\}. && \text{(3i)}
\end{aligned}$$

Starting-capacity constraints (3b) ensures that each machine k starts its processing schedule exactly once. Assignment constraints (3c) guarantees that every job j is processed exactly once on each machine k. Linking constraints (3d) links the pseudo-schedules of all machines with the global sequencing variables y_{ij}. Sequencing constraints (3e) and (3f) defines a single global permutation in which each job has one predecessor and one successor. Makespan constraint (3g) defines the makespan α as the maximum completion time of all jobs on the last machine. Domain constraints (3h) and (3i) specifies the nonnegativity and domain of the column and sequencing variables.

3.4 Position-Based Formulation

The solution method presented in this paper builds upon a mixed integer programming (MIP) formulation initially proposed by An et al. (2016) and Kim and Lee (2019). We briefly outline this formulation below, starting with the notation defined in Table 2.

The mathematical model, derived from the aforementioned formulation, is as follows:

Table 2: Notation of the Position-Based Formulation

Notation	Definition
i, j	indices of jobs, $i = 1, ..., n$, $j = 1, ..., n$.
k	indices of machines, $k = 1, ..., m$.
$[r]$	index of the job at the rth position in a (partial) schedule.
p_{ik}	processing time of job i on machine k.
x_{ir}	binary decision variable, whether job i is assigned to the rth position in a (partial) schedule.
y_{ijr}	binary decision variable, whether job i is assigned to the rth position and job j is assigned to the $(r+1)$th position.
$T_{[r]k}$	decision variable, start time of the rth job on machine k.
$C_{\max}$	decision variable, completion time of the last job on machine m.

$$
\begin{aligned}
\text{(PBF)} \quad & \min C_{\max} && && \text{(4a)}\\
& \text{s.t.} \sum_{i=1}^{n} x_{ir} = 1, && r = 1, ..., n, && \text{(4b)}\\
& \sum_{r=1}^{n} x_{ir} = 1, && i = 1, ..., n, && \text{(4c)}\\
& T_{[r]k} + \sum_{i=1}^{n} p_{ik} x_{ir} \leq T_{[r+1]k}, && r = 1, ..., n-1,\ k = 1, ..., m, && \text{(4d)}\\
& T_{[r]k} + \sum_{i=1}^{n} p_{ik} x_{ir} \leq T_{[r]\,k+1}, && r = 1, ..., n,\ k = 1, ..., m-1, && \text{(4e)}\\
& x_{ir} + x_{j(r+1)} - 1 \leq y_{ijr}, && \forall i \neq j, r = 1, 2, ..., n-1, && \text{(4f)}\\
& y_{ijr} \leq x_{ij}, && \forall i \neq j, r = 1, 2, ..., n-1, && \text{(4g)}\\
& y_{ijr} \leq x_{j(r+1)}, && \forall i \neq j, r = 1, 2, ..., n-1, && \text{(4h)}\\
& C_{\max} \geq T_{[n]m} + \sum_{i=1}^{n} p_{im} x_{in}, && && \text{(4i)}\\
& x_{ir}, y_{ijr} \in \{0, 1\}, && \forall r = 1, ..., n,\ \forall i \neq j, && \text{(4j)}\\
& T_{[r]k} \geq 0, && \forall r = 1, ..., n,\ \forall k = 1, ..., m. && \text{(4k)}
\end{aligned}
$$

The objective (4a) aims to minimize the makespan, denoted as $C_{\max}$. Constraints (4b) and (4c) ensure that each job is assigned to exactly one position, and each position is occupied by exactly one job. Constraints (4d) and (4e) enforce the fundamental logic of flow shop scheduling. Specifically, constraint (4d) ensures that jobs are assigned correct start times, such that a job at position $(r+1)$ can only begin processing after the job at position r has been completed. Constraint (4e) guarantees that jobs assigned to the same position r are processed sequentially on the machines, meaning that the $(k+1)$th machine can only start processing after the kth machine has completed. Constraints (4f)–(4h) ensure that the

relationships between variables x_{ir} and y_{ijr} are maintained, specifically that x_{ir} and its corresponding y_{ijr} share the same value. Constraint (4i) is used to calculate the maximum completion time, also known as the makespan. Constraints (4i) and (4j) enforce integrality.

3.5 Dominance Properties

Although both ATIF and PBF can solve the same flow shop scheduling problem, they provide different lower bounds when solving the linear relaxation. This difference affects the efficiency of optimization algorithms developed based on these formulations. In this section, we demonstrate the superiority of ATIF over PBF by following the approaches outlined in papers Pessoa et al. (2010) and Cire et al. (2019).

Theorem 1. For $F_m||C_{\max}$, arc-time-indexed formulation (ATIF) is at least as good as position-based formulation (PBF).

Proof: Let the set of jobs be $\mathcal{J}$, the set of position be $\mathcal{R}$, and the set of times be $\mathcal{T}$. In the model described earlier,the decision variables for PBF are x_{ir}, y_{ijr} and $T_{[r]}$. To avoid ambiguity, the decision variables for ATIF are denoted as z_{ij}^t. Next, we need to establish the mapping relationship betweem the decision variables of ATIF and PBF, i.e.,

- $z_{ij}^t = 1$, if $\exists r \in \mathcal{R}$ satisfies $x_{ir} = 1$, $t = T_{[r]}$
- $z_{ij}^t = 1$, if $y_{ijr} = 1$ and $t = T_{[r]}$
- $z_{ij}^t = 1$, if $\exists i, j \in \mathcal{J}$ satisfies $x_{ir} = 1$, $y_{ijr} = 1$, $t = T_{[r]}$

Since both ATIF and PBF aim to minimize the makespan, their objective functions are equivalent.

Theorem 2. For $F_m||C_{\max}$, arc-time-indexed formulation (ATIF) dominates position-based formulation (PBF).

Proof: In order to show that ATIF can be strictly better than PBF, define an instance of $F_m||C_{\max}$ in Table 1. The optimal solution of this instance costs 21 and has job sequence of job 2 $\rightarrow$ job 4 $\rightarrow$ job 5 $\rightarrow$ job 3 $\rightarrow$ job 1. The solution of the linear relaxation of PBF is 19.5. The optimal solution of ATIF relaxation is integral for this instance. According to Theorem 1, ATIF is as least as good as PBF. Thus ATIF dominates PBF.

4 Branch-and-Price

In this section, we develop a branch-and-price (B&P) algorithm based on the Arc-Time-Indexed Formulation (ATIF) to solve the flow shop scheduling problem. The approach combines column generation with branch-and-bound to exploit the decomposable structure of the problem. We first introduce the Restricted Master Problem (RMP) and its dual system, followed by the pricing subproblems formulated as shortest-path problems on acyclic time-expanded networks. The subsequent subsections describe the column generation process, acceleration techniques, and the integration of these components into a complete B&P framework with precedence-preserving branching rules and heuristic bounds.

4.1 Restricted Master Problem (RMP)

The column generation procedure operates on a relaxed version of the generalized set partitioning (GSP) formulation, where only a subset of columns (pseudo-schedules) is considered at each iteration.

Let $\mathcal{P}'_k \subseteq \mathcal{P}_k$ denote the current pool of generated columns for machine k. The restricted master problem (RMP) can then be formulated as follows:

$$\text{(RMP)} \quad \min_{\alpha,\lambda,y} \quad \alpha \tag{5a}$$

$$\text{s.t.} \quad (3b) - (3g), (3i) \tag{5b}$$

$$\lambda_p^k \geq 0, \qquad \forall p \in \mathcal{P}'_k, \forall k \in M. \tag{5c}$$

In this formulation, the decision variables λ_p^k represent the selection weights of the available pseudo-schedules (columns) for machine k, while y_{ij} are global linking variables enforcing the same job precedence across all machines. The makespan variable α captures the completion time on the last machine m.

At each iteration, the RMP is solved as a linear program. Dual values associated with the constraints are then used to define reduced arc costs in the pricing subproblems, where new negative-reduced-cost columns are generated and added to the RMP until optimality of the full master problem is reached.

4.2 Duals and Reduced Cost

Let $\pi_k^{(1)}$ denote the dual variable of constraint (3b) for machine k, $\pi_{jk}^{(2)}$ the dual of (3c) for job j on machine k, and $\pi_{ij}^{(3)}$ the dual of the linking constraint (3d). For a column p of machine k, i.e., a pseudo-schedule in G^k, define $q_{ijt}^{pk} \in \{0,1\}$ indicating whether arc $(i,j)^t$ on machine k is used by p. The reduced cost of column p is

$$\bar{c}(p,k) = \sum_{(i,j)^t} q_{ijt}^{p,k}\, c_{ijt}^k - \pi_k^{(1)} \sum_{(0,j)^0} q_{0j0}^{p,k} - \sum_{j\in J} \pi_{jk}^{(2)} \sum_{i\in J_0\setminus\{j\}} \sum_{t\leq T} q_{ijt}^{p,k} - \sum_{(i,j)\in\mathcal{A}} \pi_{ij}^{(3)} \sum_{t\leq T} q_{ijt}^{p,k}. \tag{6}$$

Here c_{ijt}^k is the arc cost induced by the objective terms that remain in the master (for makespan we only account cost on the last machine m). This linear arc-wise additivity of the reduced cost is the usual Dantzig-Wolfe structure; see also the arc-cost expression and shortest-path pricing in Oliveira and Pessoa (2020).

In our case, the pricing subproblem for each machine k is therefore a shortest-path problem on its time-expanded DAG G^k, with arc length equal to the corresponding reduced arc cost, a negative shortest-path value yields an entering column. This is exactly the mechanism described in the Martin-Iradi et al. (2022). The pricing problem is shortest path on a DAG, solved with dynamic programming in the linear time in the graph size.

Remark 2. (Bound equivalence) When pricing is a pure shortest-path on a DAG, the LP bound obtained by CG on the GSP equals the LP bound of the original network-flow formulation; the gain is computational (fewer active variables/constraints) rather than bound-wise. This observation and the practical speed benefits on dense graphs are documented in Martin-Iradi et al. (2022).

4.3 Column Generation

Each machine k has a directed acyclic, time-expanded network $G^k = (V^k, A^k)$ whose nodes represent feasible job-completion/start times and whose arcs encode either processing (job-start arcs) or idle

transitions. A column corresponds to a source-to-sink path (a pseudo-schedule). Shortest-path pricing on this DAG can be solved by a forward dynamic programming routine in $O(|A^k|)$; this is classical for time-indexed decompositions.

It is worth mentioning that we use multi-column pricing to accelerate the convergence of the column generation. In each iteration, we allow adding one path per machine with negative reduced cost (if any) to accelerate convergence, which is consistent with typical B&P implementations for scheduling.

For clarity, we make explicit the arc-level reduced cost used in pricing machine k. Let the last machine be m. We set each arc $(i,j)^t \in A^k$ the length:

$$\bar{c}^k_{ijt} = \mathbf{1}_{\{k=m\}}\,(t + p_{jm}) - \pi^{(1)}_k\,\mathbf{1}_{\{(i,j,t)=(0,j,0)\}} - \pi^{(2)}_{jk} - \pi^{(3)}_{ij}. \tag{7}$$

Using these reduced arc costs, the pricing subproblem for each machine k is formulated as shortest-path problem on its time-expanded directed acyclic graph G^k. Each arc;s length is set to $\bar{c}^k_{ijt}$, and the goal is to find a path (pseudo-schedule) of minimal reduced $\bar{c}(p,k)$. If the minimum reduced cost is negative, the corresponding column λ^k_p is added to the restricted master problem.

Our column generation algorithm framework is shown in Algorithm 1

Algorithm 1: Column Generation Loop

Input: Machine networks G^k for all $k \in M$; initial feasible column pool $\Lambda_0 = \{\lambda^k_p \mid k \in M\}$ (e.g., trivial "all-idle-then-one-job" pseudo-schedules).
Output: Optimal LP solution of the full master problem.
Initialize the Restricted Master Problem (RMP) with Λ_0
repeat
  Solve the RMP LP to optimality and obtain dual values $(\pi^{(1)}, \pi^{(2)}, \pi^{(3)})$
  for *each machine* $k \in M$ **do**
    Build reduced arc costs $\bar{c}^k_{ijt}$ on G^k using the duals
    Solve the DAG shortest-path problem on G^k to obtain the minimum-cost path p^* with reduced cost $\bar{c}(p^*,k)$
    if $\bar{c}(p^*,k) < 0$ **then**
      Add the corresponding column $\lambda^k_{p^*}$ to the RMP
    end
  end
until
No machine produces a negative reduced-cost column (RMP optimal for the full master).
return *Optimal RMP solution (optimal for the full master problem).*

4.4 Branch-and-Price Framework

We embed the column-generation procedure into a branch-and-price (B&P) framework to obtain integer solutions for the generalized set partitioning master problem. At each node of the search tree, we solve the node-specific restricted master problem (RMP) by column generation to near-optimality, using the resulting dual information to price new machine-wise pseudo-schedules. Problem-specific branching rules are then applied that preserve the shortest-path pricing structure on every machine network G^k.

Node Processing and Bounds. Each node ν in the B&P tree carries a set of branching decisions $\mathcal{B}_\nu$ (e.g., fixed precedences $y_{ij} = 1$ or $y_{ji} = 1$) that tighten the master constraints. The processing of node ν proceeds as follows:

1. **Warm start the RMP.** Inherit the column pool from the parent node, add the branching constraints in $\mathcal{B}_\nu$ to the RMP, and keep all previously generated columns that remain feasible.
2. **Column generation loop.** Solve the RMP to optimality; read duals $(\pi^{(1)}, \pi^{(2)}, \pi^{(3)})$. For each machine k, solve a DAG shortest-path pricing problem on G^k with reduced arc lengths $\bar{c}^k_{ijt}$. If for every k the minimum path cost is non-negative, the RMP solution is optimal for the full master at node ν.
3. **Bounding and incumbent update.** Construct a feasible integer schedule to obtain a node-specific upper bound UB_ν and update the global incumbent $UB \leftarrow \min\{UB, UB_\nu\}$.
4. **Fathoming test.** Fathom node ν if: (i) infeasible; (ii) $LB_\nu \geq UB$; or (iii) the solution is integer (both λ and y integral). Otherwise, apply branching to create child nodes and continue.

This results in a best-bound (or depth-first) B&P search. Columns are recycled between sibling nodes, and pool-purging heuristics are applied to maintain RMP compactness.

Pricing under Branching (Structure Preservation). The pricing subproblem for each machine k remains a shortest path on the acyclic time-expanded network G^k throughout the search tree. Branching decisions only add linear precedence constraints to the master (e.g., fixing y_{ij}), which modify reduced arc costs through the associated duals; The topology and feasibility of arcs $(i, j)^t \in A^k$ remain unchanged by branching, so the same forward dynamic programming routine in $O(|A^k|)$ time is reused at all nodes.

This structure-preserving property allows the B&P algorithm to maintain stable pricing times even as the tree grows.

Branching Rules. We adopt branching rules specifically designed to preserve the directed acyclic graph (DAG) shortest-path structure in the pricing problem. The primary branching mechanism is the global precedence branching, which is particularly suited for permutation flow shop problems. For any fractional pair (i, j) with $\sum_k \sum_t x^k_{ijt}$ being fractional (equivalently, fractional y_{ij}), two child nodes are created by enforcing either $y_{ij} = 1$ or $y_{ji} = 1$. This rule maintains machine-wise independence and affects only the reduced arc costs through the addition or subtraction of the dual terms $\pi^{(3)}ij$, making it natural and computationally efficient for the permutation flow shop master problem.

When the global precedence rule becomes less effective, we introduce the immediate-successor branching rule, which focuses on local sequencing decisions. Using machine-specific adjacency variables $u^kij = \sum_t x^k_{ijt}$, we branch on whether job i is the immediate predecessor of job j on machine k (i.e., $u^k_{ij} = 1$ versus$u^k_{ij} = 0$). This branching scheme is more restrictive and produces tighter bounds but is only activated when permutation-friendly branching stalls.

Finally, to further enhance search performance, we employ strong branching on ambiguous pairs. For a fractional pair (i, j) with the largest impact score—typically determined using pseudo-costs or dual violations. We temporarily evaluate both branches $y_{ij} = 1$ and $y_{ji} = 1$ through a limited number of pricing iterations to estimate the potential improvement in bounds before committing to a branching

decision. This rule is particularly beneficial near the root node, where tighter bound estimation can substantially improve convergence.

Remark 3. Branching directly on λ variables (e.g., "column p used / not used") is avoided since it destroys the decomposable structure of the master problem. Precedence-based branching ensures the shortest-path pricing remains valid and computationally efficient.

Dual Stabilization and Convergence Aids. Plain column generation often suffers from oscillations in the dual variables, which can result in unstable or slow convergence. To address this issue, we incorporate several stabilization strategies. First, we adopt a box-constrained dual approach (also known as bundle stabilization), where the dual variables are maintained within a trust region around a moving center, with penalties imposed for deviations to dampen fluctuations. We also use interior-point warm starts by performing a few barrier iterations to obtain smoother dual solutions before transitioning to simplex-based column generation. Additionally, multiple columns are allowed to enter the basis per iteration, typically one per machine or the top r negative-cost paths, which helps to accelerate convergence. Finally, we employ an early stopping criterion based on ε-optimality: the pricing routine terminates once all reduced costs satisfy $\bar{c}^k_{ijt} \geq -\varepsilon$, allowing small violations to improve computational efficiency without significantly affecting bound quality.

Primal Heuristics (Upper Bounds). High-quality feasible solutions can significantly accelerate the convergence of branch-and-price by tightening upper bounds early in the search. To this end, we incorporate two primal heuristics. The first is a rounding-and-repair strategy, where a tentative permutation is extracted from the fractional y_{ij} values, typically by sorting jobs based on topological scores, and earliest feasible start times are computed to construct a valid schedule. The second approach involves a diving procedure, in which a subset of strongly fractional precedences (e.g., those with $y_{ij} \approx 1$) is temporarily fixed, and a short column generation phase is executed to complete the remaining sequence. Among the resulting solutions, the best is retained. The corresponding upper bounds UB_ν obtained through these heuristics are recorded and used to prune nodes via bound dominance.

Reduced Costs at Node ν. At node ν, with RMP duals $(\pi^{(1)}, \pi^{(2)}, \pi^{(3)})$, the reduced arc cost on machine k is given by equation 7. This expression is identical to the one used in the root node, ensuring that branching only modifies the dual-related cost shifts without altering the network topology or the shortest-path structure.

5 Branch-and-Bound

In this section, we outline the branch-and-bound (B&B) framework commonly used in the literature to obtain exact solutions for flow shop scheduling problems (e.g., Yang and Chern (1995), An et al. (2016), Kim and Lee (2019)). We begin by introducing two classical constructive heuristics (Johnson's algorithm and the NEH algorithm) along with our proposed greedy heuristic. These methods are used to generate high-quality initial solutions that serve as upper bounds in the B&B process. We then describe the key components of the B&B framework used for performance comparison in this paper, including the branching strategy, bounding mechanisms, and search procedure, which collectively influence the algorithm's efficiency and convergence.

5.1 Heuristic Initial Solution

To enhance the efficiency of the branch-and-bound framework, we utilize three heuristic methods for initial solution generation as the upper bounds. Modified Johnson's algorithm and the NEH algorithm are adapted from existing literature, while the third is a newly designed greedy algorithm. These methods serve as initialization procedures for the branch-and-bound approach detailed in the next subsection.

5.1.1 Modified Johnson's Algorithm

Johnson's algorithm is an optimal scheduling rule for the two-machine flowshop problem (Johnson, 1954). However, when the number of machines exceeds two, Johnson's algorithm can no longer provide an optimal solution. In this study, we employ a generalized version of Johnson's algorithm, known as the Campbell–Dudek–Smith (CDS) method (Campbell et al., 1970), to generate good approximate solutions for the m-machine flowshop problem. The basic idea of this method is to transform the m-machine problem into a series of $(m-1)$ equivalent two-machine subproblems so that Johnson's rule can still be applied.

For each subproblem $k = 1, 2, \ldots, m-1$, the processing time of each job on the first virtual machine is defined as the sum of its processing times on the first k machines, while the processing time on the second virtual machine is defined as the sum of its processing times on the remaining $(m-k)$ machines. Then, Johnson's rule is applied to each two-machine subproblem to obtain a job sequence that minimizes the makespan in the corresponding simplified system. The makespan of each candidate sequence is subsequently evaluated in the real m-machine environment, and the sequence producing the smallest makespan is selected as the final schedule.

The detailed procedure of the Modified Johnson's algorithm is summarized in Algorithm 2.

Algorithm 2: Modified Johnson's Algorithm

Input: n jobs, m machines; processing times $p_{i,k}$ for job i on machine k
Output: A job sequence π
$BestSequence \leftarrow \varnothing$, $BestC_{\max} \leftarrow +\infty$
for $k = 1$ **to** $m-1$ **do**
  for *each job* i **do**
    $P1_i \leftarrow \sum_{t=1}^{k} p_{i,t}$
    $P2_i \leftarrow \sum_{t=k+1}^{m} p_{i,t}$
  end
  $\pi^{(k)} \leftarrow \text{JohnsonSort}(P1, P2)$
  Compute $C_{\max}^{(k)}$ for $\pi^{(k)}$ on the real m-machine system
  if $C_{\max}^{(k)} < BestC_{\max}$ **then**
    $BestC_{\max} \leftarrow C_{\max}^{(k)}$
    $BestSequence \leftarrow \pi^{(k)}$
  end
end
return $BestSequence$

5.1.2 Modified NEH Algorithm

The NEH algorithm proposed by Nawaz et al. (1983) is one of the most effective constructive heuristics for the m-machine permutation flowshop scheduling problem. In this algorithm, jobs are first sorted

in non-increasing order of their total processing times across all machines. Then, the schedule is built iteratively: each job is inserted into every possible position in the current partial sequence, and the position that yields the smallest makespan is retained. This process continues until all jobs are scheduled, producing a near-optimal permutation with low computational effort and high solution quality.

The detailed procedure of the NEH algorithm is summarized in Algorithm 3.

Algorithm 3: NEH Algorithm

Input: n jobs, m machines; processing times $p_{i,k}$ for job i on machine k
Output: A job sequence S
for *each job* i **do**
  $T_i \leftarrow \sum_{k=1}^{m} p_{i,k}$
end
Sort jobs in non-increasing order of T_i to obtain $\pi = (i_1, i_2, \ldots, i_n)$
$S \leftarrow (i_1)$
for $j = 2$ **to** n **do**
  $bestC_{\max} \leftarrow +\infty$
  for $pos = 1$ **to** $|S| + 1$ **do**
    $S' \leftarrow$ Insert i_j into position pos of S
    $C_{\max} \leftarrow \texttt{ComputeMakespan}(S', p)$
    if $C_{\max} < bestC_{\max}$ **then**
      $bestC_{\max} \leftarrow C_{\max}$
      $bestSequence \leftarrow S'$
    end
  end
  $S \leftarrow bestSequence$
end
return S

5.1.3 Greedy Algorithm

The proposed greedy algorithm constructs a job sequence iteratively by appending the job that causes the smallest incremental increase in the estimated makespan at each step. Let S denote the current partial sequence and U the set of unscheduled jobs. For each candidate job $j \in U$, we define the incremental cost of placing j immediately after the last job $u = \text{last}(S)$ as

$$\Delta(j \mid S) = \sum_{k=1}^{m} \max\{0,\, p_{j,k} - p_{u,k}\}, \tag{8}$$

where $p_{i,k}$ is the processing time of job i on machine k.

This approximation reflects the additional load that job j would introduce on each machine relative to its predecessor u; the larger the difference, the higher the delay propagated through the flowshop.

At each iteration, the algorithm selects the job $j^\star = \arg\min_{j \in U} \Delta(j \mid S)$, and appends it to the sequence S. This process repeats until all jobs are scheduled. Finally, the resulting sequence $S = (S_1, S_2, \ldots, S_n)$ is evaluated on the real m-machine system using the exact makespan function.

By minimizing the incremental propagation of processing load across machines, this greedy rule effectively maintains a balanced utilization of all stages and achieves near-optimal schedules with very low computational complexity.

The detailed procedure of the greedy algorithm is summarized in Algorithm 4.

Algorithm 4: Greedy Algorithm

Input: n jobs, m machines; processing times $p_{i,k}$
Output: A job sequence S
$S \leftarrow (), \quad U \leftarrow \{1, 2, \ldots, n\}$
$i^\star \leftarrow \arg\max_{i \in U} \sum_{k=1}^{m} p_{i,k}$
$S \leftarrow (i^\star), \quad U \leftarrow U \setminus \{i^\star\}$
while $U \neq \emptyset$ **do**
  $best \leftarrow +\infty, \quad j^\star \leftarrow$ null
  foreach $j \in U$ **do**
    $\Delta \leftarrow \texttt{IncrementalCostApprox}(S, j, p)$
    if $\Delta < best$ **then**
      $best \leftarrow \Delta, \quad j^\star \leftarrow j$
    end
  end
  $S \leftarrow S \oplus j^\star, \quad U \leftarrow U \setminus \{j^\star\}$
end
return S

5.2 Branch-and-Bound Framework

Before describing the branch-and-bound procedure, we define the concept of the partial schedule in definition 5.2.

Definition 2. (Partial Schedule) A *partial schedule* $\sigma = ([1], [2], \ldots, [r])$ represents the ordered sequence of the first r jobs assigned at the front of a complete permutation schedule, where $r \leq n$. The remaining unscheduled jobs form the set $U = N \setminus \{[1], \ldots, [r]\}$, and $C_k(\sigma)$ denotes the completion time of all jobs in σ on machine k.

Before generating nodes in the B&B tree, an initial upper bound UB is first obtained using the heuristic algorithms introduced in the previous section (Johnson, NEH, and the greedy rule). Among the schedules generated by these heuristics, the one with the minimum makespan is selected as the initial upper bound UB_0. The search then proceeds recursively in a depth-first manner, using the framework described below.

Node Representation. Each node in the B&B tree corresponds to a partial schedule, denoted by $\sigma = ([1], [2], \ldots, [r])$, where r jobs have been sequenced and $n - r$ jobs remain unscheduled. The completion times on the m machines for this partial schedule are computed recursively as:

$$C_{[i],1} = C_{[i-1],1} + p_{[i],1}, \; C_{[i],k} = \max(C_{[i],k-1}, \, C_{[i-1],k}) + p_{J_{[i]},k}, \; k = 2, \ldots, m. \tag{9}$$

The corresponding makespan of the partial schedule is $C_{\max}(\pi) = C_{[r],m}$.

Branching Strategy. From a node $\sigma = (J_{[1]}, \ldots, J_{[r]})$, the algorithm generates $n - r$ child nodes by appending one unscheduled job $j \in N \setminus \sigma$ to the end of the partial sequence:

$$\sigma' = (\sigma, j) = ([1], \ldots, [r], j) \tag{10}$$

Bounding Procedure. or each node σ, a lower bound $LB(\sigma)$ on the makespan is computed to assess whether the node may contain an optimal solution. The lower bound is derived from the completion times of the partial schedule on the first r jobs and the remaining total processing time of unscheduled jobs. Let $R = N \setminus \sigma$ denote the set of unscheduled jobs, then:

$$LB(\pi) = \max_{k=1,\ldots,m} \left(C_{[r],k} + \sum_{t=k}^{m} \min_{j \in R} p_{j,t} \right). \tag{11}$$

This bound represents the earliest possible completion time assuming that, for each machine k, the remaining operations are processed in the most favorable order.

A node is pruned if $LB(\pi) \geq UB$, since no better solution can be obtained from this branch.

Upper Bound Update. Whenever a complete sequence σ (of length n) is generated, its exact makespan $C_{\max}(\sigma)$ is computed. If $C_{\max}(\pi) < UB$, the upper bound is updated as $UB \leftarrow C_{\max}(\pi)$, and the current sequence σ^* is stored as the best solution.

Node Selection and Search Strategy. At each iteration, a node is selected for branching. The algorithm employs a depth-first rule: among all active (unfathomed) nodes, the one corresponding to the longest partial sequence (i.e., the largest r) is selected first. In case of ties, the node with the smallest lower bound $LB(\sigma)$ is preferred. This rule allows early detection of complete feasible solutions, which helps in tightening the upper bound and increasing pruning efficiency.

Dominance and Fathoming. Two dominance rules are applied to reduce redundant search:

1. If two partial schedules σ_1 and σ_2 yield identical sets of scheduled jobs and $C_{[r],k}(\sigma_1) \leq C_{[r],k}(\sigma_2)$ for all k, then σ_2 is dominated and discarded.
2. Any node whose lower bound equals or exceeds the current best UB is fathomed.

The proposed algorithm is illustrated using the data from Example 3.1 originally presented in (Yang and Chern, 1995), as shown in Figure 3.

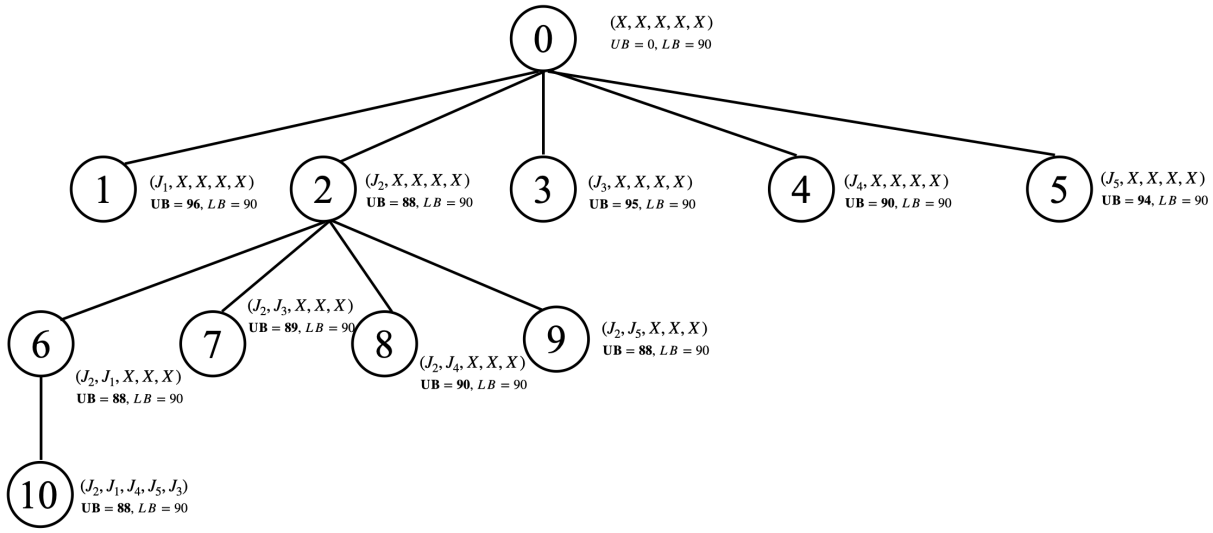

Figure 3: The branch-and-bound tree of example 1

6 Computational Experiments

This section presents a comprehensive evaluation of the computational performance of the proposed Arc-Time-Indexed Formulation (ATIF) and its corresponding Branch-and-Price (B&P) solution algorithm. To benchmark its efficiency and effectiveness, we conduct comparative experiments against the traditional Position-Based Formulation (PBF) solved using a Branch-and-Bound (B&B) approach, as well as three widely recognized heuristics: the Modified Johnson's Algorithm, the NEH Heuristic, and the Greedy Algorithm. All computational experiments were performed on a MacBook Pro equipped with an Apple M2 chip (8-core CPU), 8 GB of unified memory, and 256 GB SSD, running macOS Ventura 13.x. The implementation was carried out in Python 3.12.7, using PyCharm 2024.3.4 (Community Edition) as the integrated development environment, and Gurobi 11.0.3 as the optimization solver.

The computational assessment covers all combinations of job sizes $n \in \{10, 15, 20, 25, 30, 35, 40, 45, 50\}$ and machine counts $m \in \{2, 3, 5\}$. The first three experimental blocks model regular instances: every processing time is drawn from the discrete uniform distribution $U[5, 10]$. The fourth block introduces extreme outliers: for each instance, 80% of the operations retain normal times $U[5, 10]$, whereas the remaining 20% are designated outliers and assigned dramatically longer durations from $U[50, 60]$. Twenty independent replicates are generated for every (n, m) pair. Time limits are set to 3600s for the three regular blocks and reduced to 30s for the outlier block, so as to probe both scalability and robustness under processing-time shocks.

6.1 Test of the Heuristic Algorithms

We first assess the effectiveness of three constructive heuristics used as initial upper bounds: Modified Johnson's Algorithm (MJ), Greedy Algorithm (GR), and the classical NEH Algorithm (NEH). Following the convention in related flowshop studies, we report the average percentage gap (Avg gap) and the number of best solutions (NBS), aggregated by instance groups, together with the average CPU time and cross-algorithm time ratios (ARCPUT); see Table 3.

Because optimal solutions of our synthesized instances are unknown, the gap is computed with respect to the best makespan obtained among the three heuristics for each instance (see the table note). This "best-of-heuristics" yardstick and the NBS count are standard in comparative studies of constructive schedules for constrained flow shops.

Over all 200 instances (10 groups $\times$ 20 instances), GR attains the smallest overall average gap, 2.30% and the largest NBS $= 112$. NEH follows closely with 2.79% and NBS $= 80$, while MJ is clearly dominated in solution quality with 11.99% and NBS $= 8$. For $m = 3$, both GR and NEH are robust as n increases: GR's Avg gaps remain near 2–3% and NEH near 2–4% across (20,3)–(50,3). In contrast, MJ's Avg gap grows steeply. For $m = 5$, the picture is similar. GR keeps Avg gaps around 3% (2.67–3.41%), and NEH stays competitive, occasionally edging GR, for (50,5) NEH posts the best Avg gap (2.01%). MJ again degrades markedly with size.

Takeaways. (i) GR is the most reliable "default" initializer in our setting—best overall Avg gap and the highest NBS while also being time-efficient. (ii) NEH is a strong second choice and occasionally dominates on larger, tighter cases (e.g., n=50, m=5). (iii) MJ is not competitive despite its simplicity. Consequently, in the subsequent branch-and-bound framework we adopt GR as the primary initializer.

6.2 Test of the Formulation Strength

We next compare the model strength of two formulations: the classical Position-Based Formulation (PBF) and the proposed Arc-Time-Indexed Formulation (ATIF). Both models are solved directly by the Gurobi. We report the average optimality gap (Avg gap), average CPU time (Avg time), and maximum CPU time (Max time) for each group of instances; see Table 4.

Because every instance can be solved to proven optimality by the ATIF model, the optimal solutions obtained by ATIF are used as the reference values for computing the solver-reported gaps of the PBF model. The CPU time of unsolved PBF cases is truncated at 3600s, following the standard practice adopted in computational comparisons of model strength.

Across all instance groups, ATIF consistently dominates PBF in both efficiency and stability. For small-sized problems (e.g., (10, 2) and (10, 3)), both models solve to optimality almost instantly. However, as the problem scale increases, the performance difference grows sharply. For medium-sized groups (e.g., (13, 5), (15, 3), and (15, 5)), ATIF requires only 10–20% of the CPU time consumed by PBF, while maintaining identical optimal gaps (0 %). In contrast, PBF's runtime increases rapidly with m, and its solver often reaches the 3600s limit for larger cases.

For the most challenging configurations (18, 5) and (20, 5), PBF fails to reach optimality within the time limit, leaving residual gaps of 4.98% and 12.07%, respectively. ATIF, on the other hand, successfully solves all instances to optimality within reasonable time, on average 675s for (18, 5) and 1199s for (20, 5). These results demonstrate that the linear relaxation of ATIF provides much tighter bounds, leading to faster convergence of the solver.

Takeaways. (i) ATIF yields identical or better solution quality for every tested instance. (ii) Its computational time grows far more slowly with instance size, showing a clear scalability advantage. (iii) The superior relaxation tightness of ATIF allows the solver to prune the branch-and-bound tree more efficiently. Consequently, in the following subsection 6.3 we employ ATIF as the default master formulation in our exact algorithms.

Table 3: Results of the test on the effectiveness of the heuristics

	MJ[1]			GR[2]			NEH[3]			ARCPUT[4]	
(n, m)	Avg gap[5] (%)	Avg time (s)	NBS[6]	Avg gap (%)	Avg time (s)	NBS	Avg gap (%)	Avg time (s)	NBS	$T_{\text{GR}}/T_{\text{MJ}}$	$T_{\text{NEH}}/T_{\text{MJ}}$
(10,3)	0.15	0.01	3	0.02	0.00	16	0.10	0.00	1	0.00	0.00
(10,5)	0.55	0.98	4	0.83	1.39	12	0.48	0.80	4	1.42	0.82
(20,3)	4.73	1.47	1	2.95	0.95	10	3.43	1.12	9	0.64	0.76
(20,5)	10.21	14.33	0	2.67	3.00	12	4.55	3.42	8	0.21	0.32
(30,3)	17.14	4.41	0	2.41	1.91	13	4.12	3.44	7	0.43	0.78
(30,5)	13.59	10.05	0	3.01	6.94	12	4.01	4.09	8	0.69	0.41
(40,3)	19.88	6.09	0	2.09	4.38	10	2.62	6.58	10	0.72	1.08
(40,5)	17.35	32.14	0	2.98	10.45	11	4.10	14.76	9	0.33	0.46
(50,3)	19.02	21.65	0	2.65	7.87	7	2.44	10.12	13	0.36	0.47
(50,5)	17.30	52.00	0	3.41	16.39	9	2.01	33.79	11	0.32	0.65
Overall	11.99	-	8	2.30	-	112	2.79	-	80	-	-

Note: There are 20 instances for each group size. Since the Johnson's algorithm can find the optimal solution for the flow-shop scheduling problem with 2 machines, this table only includes instances with 3 and 5 machines.

[a] Modified Johnson's algorithm.

[b] Greedy algorithm.

[c] NEH algorithm.

[d] Average ratio of CPU times.

[e] Since the optimal solutions of the synthesized instances are unknown, the gap here is calculated by taking the best solution among the three heuristic algorithms as the benchmark, and then measuring the gap of other heuristic solutions relative to this solution, i.e., gap $= (T_{MJ} - \min\{T_{MJ}, T_{MG}, T_{MN}\})/T_{MJ}$, where the notation T denotes the completion time obtained by the current algorithm. The calculation method for the other two is the same.

[f] Number of instances (out of 20 instances) for which the algorithm found the best solution among the three heuristic algorithms.

Table 4: Results of the test on model strength between ATIF and PBF

	PBF-G[1]			ATIF-G[2]		
(n, m)	Avg gap[3]	Avg time[4]	Max time	Avg gap	Avg time	Max time
(10,2)	0.00	0.00	0.00	0.00	0.01	0.03
(10,3)	0.00	0.07	0.13	0.00	0.06	0.13
(10,5)	0.00	23.31	34.11	0.00	3.17	4.12
(13,2)	0.00	4.72	6.29	0.00	0.01	0.03
(13,3)	0.00	9.45	13.33	0.00	4.42	6.13
(13,5)	0.00	179.87	298.54	0.00	19.68	24.39
(15,2)	0.00	5.39	6.21	0.00	1.17	1.89
(15,3)	0.00	19.77	23.02	0.00	6.82	10.30
(15,5)	0.00	787.10	1311.00	0.00	122.09	168.09
(18,2)	0.00	7.22	10.09	0.00	2.57	4.33
(18,3)	0.00	69.86	97.85	0.00	27.65	35.58
(18,5)	4.98	$\geq$3600	$\geq$3600	0.00	675.32	987.89
(20,2)	0.00	11.39	14.21	0.00	4.41	6.55
(20,3)	0.00	197.03	231.49	0.00	89.43	126.93
(20,5)	12.07	$\geq$3600	$\geq$3600	0.00	1198.79	1655.71

Note: There are 20 instances for each group size.

[a]Solve the PBF model directly by using the Gurobi solver.

[b]Solve the ATIF model directly by using the Gurobi solver.

[c]Since every instance can be solved to optimality by directly invoking the solver on the ATIF formulation, the optimality gap of the PBF model returned by the solver is computed as gap $= (T - T_{\text{opt}})/T$, where the notation T denotes the completion time obtained by the current algorithm.

[d]Average CPU time (or its lower bound), obtained by assigning a value of 3600s to any instance not solved to optimality within that limit.

6.3 Test of the Exact Algorithms

We then compare the computational performance of two exact algorithms: Branch-and-Bound on the Position-Based Formulation (PBF-BB) and Branch-and-Price on the Arc-Time-Indexed Formulation (ATIF-BP). Both algorithms use identical hardware and termination criteria, with a 3600s time limit per instance. The initial feasible solutions of PBF-BB are generated by the modified greedy heuristic, while The initial feasible solutions of ATIF-BP are obtained through a dynamic programming procedure on the time-expanded acyclic network. The key performance indicators include average optimality gap (Avg gap), average and maximum CPU time, and number of unsolved instances; see Table 5.

Because the true optimal values of large-scale instances are unknown, the gaps of both algorithms are measured by the relative distance between their upper and lower bounds. This uniform criterion enables a fair comparison of the solution quality and convergence behavior.

Across all instance groups, ATIF-BP clearly outperforms PBF-BB in both computational efficiency and robustness. For small instances (e.g., (10, 2)–(20, 3)), both algorithms reach optimality within seconds. However, as the problem size increases, the performance gap expands sharply. For example, at (25, 5), ATIF-BP solves all 20 instances within 18s on average, whereas PBF-BB requires more than 200s; at (30, 5), ATIF-BP achieves optimality within 30s while PBF-BB consumes nearly 1900s and already fails to solve two instances within the limit.

For larger configurations, the advantage of ATIF-BP becomes even more pronounced. When $n \geq 35$ and $m \geq 3$, PBF-BB frequently fails to close the gap before timeout and up to 20 unsolved cases. In contrast, ATIF-BP maintains zero gaps for nearly all instances, except for the extremely large case (50, 5), where it still achieves an average gap of 13.22% while PBF-BB remains entirely unsolved. Moreover, the CPU time of ATIF-BP scales much more gently with n and m.

Takeaways. (i) ATIF-BP consistently achieves smaller gaps and faster convergence across all instance sizes. (ii) Its computational time grows much more slowly, demonstrating strong scalability. (iii) The superior relaxation strength and decomposition design of ATIF make it significantly more effective for medium and large-scale flow-shop scheduling.

6.4 Test of Formulation Robustness under Extreme Processing Time

In industrial production systems, it is quite common that certain machines act as bottlenecks, where the processing times of jobs are significantly longer than on other machines due to limited capacity or specialized operations. To assess the robustness and adaptability of our proposed approach under these realistic conditions, we conduct a numerical experiment that emulates various bottleneck job distributions.

Three processing-time distribution patterns are considered:

- UNI, a purely uniform distribution in [5,10];
- MIX, a mixed distribution with 90% of jobs drawn from $U[5, 10]$ and 10% from $U[50, 60]$;
- FIX, a fixed bottleneck machine where processing times are drawn from $U[50, 60]$ while the remaining machines follow $U[5, 10]$.

Both the Branch-and-Bound on PBF (PBF-BB) and the Branch-and-Price on ATIF (ATIF-BP) are tested under a 30s time limit per instance. The performance indicators include the average optimality

Table 5: Results of the test on performance comparison between B&B on PBF and B&P on ATIF

	PBF-BB[1]				ATIF-BP[2]			
(n, m)	Avg gap[3] (%)	Avg time[4] (s)	Max time (s)	Unsolved[5]	Avg gap (%)	Avg time (s)	Max time (s)	Unsolved
(10,2)	0.00	0.00	0.00	0	0.00	0.00	0.00	0
(10,3)	0.00	0.02	0.03	0	0.00	0.00	0.01	0
(10,5)	0.00	3.19	5.00	0	0.00	0.67	0.77	0
(15,2)	0.00	0.94	1.13	0	0.00	0.01	0.02	0
(15,3)	0.00	1.57	1.99	0	0.00	0.01	0.02	0
(15,5)	0.00	17.20	19.24	0	0.00	1.32	1.57	0
(20,2)	0.00	1.38	1.67	0	0.00	0.11	0.16	0
(20,3)	0.00	7.80	10.02	0	0.00	1.16	1.57	0
(20,5)	0.00	98.10	134.99	0	0.00	2.78	3.02	0
(25,2)	0.00	8.22	11.21	0	0.00	0.65	0.98	0
(25,3)	0.00	39.92	48.67	0	0.00	9.25	10.04	0
(25,5)	0.00	249.88	288.75	0	0.00	15.21	17.92	0
(30,2)	0.00	16.90	18.90	0	0.00	1.99	3.02	0
(30,3)	0.00	127.03	155.09	0	0.00	10.12	13.34	0
(30,5)	1.19	1906.58	2328.96	2	0.00	24.33	30.53	0
(35,2)	0.00	21.30	23.49	0	0.00	4.88	5.01	0
(35,3)	0.00	472.03	500.00	0	0.00	37.09	43.01	0
(35,5)	9.89	$\geq$3600	≥ 3600	9	0.00	68.35	81.09	0
(40,2)	0.00	43.21	49.11	0	0.00	11.12	15.58	0
(40,3)	8.91	2662.76	≥ 3600	6	0.00	96.32	133.65	0
(40,5)	15.34	$\geq$3600	≥ 3600	11	0.00	128.09	142.20	0
(45,2)	0.00	139.56	176.89	0	0.00	16.59	21.54	0
(45,3)	23.75	≥ 3600	≥ 3600	17	0.00	232.55	265.22	0
(45,5)	26.65	$\geq$3600	≥ 3600	18	0.00	657.76	743.09	0
(50,2)	0.00	651.31	883.65	0	0.00	26.11	31.00	0
(50,3)	39.98	≥ 3600	≥ 3600	20	0.00	1112.92	1410.83	0
(50,5)	40.05	$\geq$3600	≥ 3600	20	13.22	≥ 3600	≥ 3600	9

Note: There are 20 instances for each group size.

[a]Solve the PBF model by branch-and-bound algorithm. Initial solutions are obtained via the modified greedy algorithm.

[b]Solve the ATIF model by branch-and-price algorithm. Initial solutions are obtained via the Dijkstra's algorithm in the network graph.

[c]Since the optimal values of many instances are unknown, the gaps of both algorithms are uniformly measured by the relative distance between their upper and lower bounds, i.e., gap $=$ (UB $-$ LB)/UB, where the notation UB denotes upper bound, and the notation LB denotes lower bound.

[d]Average CPU time (or its lower bound), obtained by assigning a value of 3600s to any instance not solved to optimality within that limit.

[e]The Number of instances among 20 instances that have not been solved to optimality within 3600s.

gap (Avg gap), average CPU time (Avg time), and the coefficients of variation of both metrics (CVG and CVT); see Table 6.

Because the true optimal values of these synthetic instances are unknown, all gaps are computed from the relative distance between solver-reported upper and lower bounds, ensuring consistent comparison across both algorithms.

Across all distributions, ATIF-BP exhibits remarkable robustness, maintaining zero gaps and near-constant computation times across all problem sizes and bottleneck intensities. For example, in the severe FIX setting at (20,5), ATIF-BP solves all instances within 5s on average, whereas PBF-BB fails to converge within the 30s cap, showing an average gap of 17.9%. Even for moderate configurations such as (15,5) and (20,3), ATIF-BP remains stable with negligible variability (CVG ≈ 0, CVT < 0.2), while PBF-BB experiences sharp fluctuations in both optimality and runtime.

Under the MIX and FIX distributions, where a small portion or single machine acts as a bottleneck, the gap and time variability of PBF-BB increase significantly (e.g., CVG up to 0.6 and CVT > 0.7 for (20,5)), reflecting its sensitivity to skewed processing structures. By contrast, ATIF-BP's performance remains almost unaffected, confirming the stability of its decomposition framework and its insensitivity to heterogeneity in job durations.

Takeaways. (i) ATIF-BP demonstrates superior robustness across uniform and highly unbalanced processing conditions. (ii) The coefficients of variation of both gap and runtime remain extremely low, confirming its stable convergence behavior. (iii) PBF-BB is highly sensitive to bottlenecks, with degraded convergence and time stability as the imbalance grows.

Table 6: Results of the test on robustness under bottleneck job conditions

		PBF-BB[1]				ATIF-BP[2]			
(n, m)	PDS[3]	Avg gap[4]	Avg time[5]	CVG[6]	CVT[7]	Avg gap	Avg time	CVG	CVT
(10,3)	UNI	0.00	0.03	0.00	0.11	0.00	0.00	0.00	0.01
(10,3)	MIX	0.00	0.02	0.00	0.12	0.00	0.09	0.00	0.02
(10,3)	FIX	0.00	0.03	0.00	0.09	0.00	0.06	0.00	0.01
(10,5)	UNI	0.00	3.77	0.00	0.12	0.00	0.62	0.00	0.07
(10,5)	MIX	0.00	3.98	0.00	0.12	0.00	1.45	0.00	0.08
(10,5)	FIX	0.00	4.01	0.00	0.13	0.00	1.89	0.00	0.08
(15,3)	UNI	0.00	1.06	0.00	0.63	0.00	0.01	0.00	0.10
(15,3)	MIX	0.00	1.65	0.00	0.21	0.00	0.22	0.00	0.19
(15,3)	FIX	0.00	1.62	0.00	0.22	0.00	0.81	0.00	0.16
(15,5)	UNI	1.00	18.05	0.05	0.10	0.00	1.58	0.00	0.12
(15,5)	MIX	3.32	23.74	0.39	0.25	0.00	2.31	0.00	0.10
(15,5)	FIX	4.54	26.65	0.27	0.71	0.00	3.02	0.00	0.19
(20,3)	UNI	0.00	7.42	0.00	0.15	0.00	1.17	0.00	0.15
(20,3)	MIX	0.00	7.88	0.00	0.20	0.00	1.99	0.00	0.13
(20,3)	FIX	0.00	8.94	0.00	0.28	0.00	1.85	0.00	0.17
(20,5)	UNI	17.44	$\geq$30.00	0.61	0.30	0.00	2.61	0.00	0.18
(20,5)	MIX	16.53	$\geq$30.00	0.57	0.69	0.00	6.45	0.00	0.20
(20,5)	FIX	17.92	$\geq$30.00	0.62	0.24	0.00	5.04	0.00	0.18

Note: There are 20 instances for each group size. The upper limit of the solution time is set at 30 seconds.

[a]Solve the PBF model by branch-and-bound algorithm. Initial solutions are obtained via the modified greedy algorithm.

[b]Solve the ATIF model by branch-and-price algorithm. Initial solutions are obtained via the Dijkstra's algorithm in the network graph.

[c]The processing time distribution pattern. UNI for purely uniform processing times in [5,10]; MIX for a mixed distribution with 90% in U[5,10] and 10% in U[50,60]; FIX for a fixed bottleneck machine where processing times are in U[50,60] and others in U[5,10].

[d]The gaps of both algorithms are uniformly measured by the relative distance between their upper and lower bounds, i.e., gap $=$ (UB $-$ LB)/UB, where the notation UB denotes upper bound, and the notation LB denotes lower bound.

[e]Average CPU time (or its lower bound), obtained by assigning a value of 3600s to any instance not solved to optimality within that limit.

[f]Coefficient of Variation of the optimality Gap.

[g]Coefficient of Variation of the solve Time.

7 Conclusions and Future Work

In this section, we conclude the study by summarizing the main theoretical and computational contributions of the proposed Arc-Time-Indexed Formulation (ATIF) and Branch-and-Price (B&P) framework, and by outlining several directions for future work. The first part presents key conclusions and insights from our modeling and algorithmic results, while the second part discusses potential extensions and hybrid approaches for improving scalability and applicability in real-world scheduling environments.

7.1 Conclusions

This paper investigates the classical permutation flow shop scheduling problem ($F_m||C_{\max}$), which is widely recognized for its theoretical complexity and practical relevance in industrial production systems. While existing exact methods such as the Position-Based Formulation (PBF) have been shown to work well for small-scale instances, they often become computationally intractable as the size of the instance or the number of machines increases. To overcome these limitations, we propose a novel modeling and algorithmic framework based on an extended Arc-Time-Indexed Formulation (ATIF) and an exact Branch-and-Price (B&P) algorithm, designed specifically for the flow shop environment.

The core contributions of this work are threefold. First, we introduce the ATIF to model the flow shop scheduling problem by constructing a time-expanded network graph. Compared to PBF, ATIF provides a tighter linear relaxation and captures job sequencing through arc flows, making it better suited for decomposition and exact optimization. Furthermore, we modified the standard arc-time-indexed network by introducing idle arcs and virtual initial arcs, thereby preserving the feasibility and logical flow of job processing across multiple machines.

Second, building upon the ATIF structure, we design an exact Branch-and-Price (B&P) algorithm that solves a generalized set partitioning reformulation of the problem. The algorithm incorporates a shortest-path-based pricing subproblem embedded within a column generation framework, along with tailored branching strategies that preserve the structure of the pricing problem. To enhance convergence, we implement dual stabilization techniques and primal heuristics for upper bound tightening. This integrated design enables the efficient solution of medium- to large-scale instances with provable optimality.

Third, the experiments reveal that the number of machines plays a critical role in determining the computational complexity of the problem. While increasing the number of jobs causes a linear growth in complexity, increasing the number of machines introduces exponential growth in feasible sequencing combinations. This suggests that machine dimension is the dominant factor in the difficulty of solving flow shop instances, and emphasizes the importance of model scalability in this dimension.

From a practical perspective, our proposed method offers a promising exact solution approach for medium-scale flow shop scheduling problems, especially in industries such as semiconductor manufacturing, automotive assembly, and printing, where makespan minimization is crucial for operational efficiency. The strong performance of our method also opens up opportunities for embedding it into larger integrated scheduling systems, where flow shop scheduling is a subcomponent of broader decision-making problems.

7.2 Future work

Despite its strengths, our current study also leaves several open directions for future research. One immediate extension is to generalize the ATIF and B&P framework to accommodate real-world constraints such as sequence-dependent setup times, limited waiting times, blocking constraints, and machine el-

igibility restrictions. These additions would significantly enhance the applicability of our method in realistic industrial settings but would also require new adjustments to the network structure and pricing subproblems.

Another direction lies in the hybridization of our exact approach with heuristic or metaheuristic methods. While exact algorithms guarantee optimality, they may still become computationally expensive for very large instances. Developing hybrid solvers that combine the robustness of B&P with the speed of heuristics, such as Large Neighborhood Search or Genetic Algorithms, could help to scale the method to larger problem sizes without sacrificing too much solution quality.

Finally, there is increasing interest in the application of learning-based methods to accelerate exact optimization. One promising avenue is to use machine learning models to predict good initial columns for the column generation process or to guide branching decisions in the B&P tree. This could help to reduce the number of iterations and speed up convergence without affecting optimality guarantees.

In summary, this research lays the foundation for a new line of work on network-based exact formulations for flow shop scheduling. The extended Arc-Time-Indexed Formulation (ATIF) and the Branch-and-Price (B&P) algorithm presented in this paper jointly offer a powerful and flexible toolset for solving complex scheduling problems with theoretical rigor and practical efficiency.

Declarations

Conflicts of interest The authors declare no conflict of interest.

References

Young-Jin An, Yeong-Dae Kim, and Seong-Woo Choi. Minimizing makespan in a two-machine flowshop with a limited waiting time constraint and sequence-dependent setup times. *Computers & Operations Research*, 71:127–136, 2016.

Pasquale Avella, Maurizio Boccia, and Bernardo D'Auria. Near-Optimal Solutions of Large-Scale Single-Machine Scheduling Problems. *INFORMS Journal on Computing*, 17:183–191, 2005.

Pasquale Avella, Maurizio Boccia, Carlo Mannino, and Igor Vasilyev. Time-indexed formulations for the runway scheduling problem. *Transportation Science*, 51(4):1196–1209, 2017.

Louis-Philippe Bigras, Michel Gamache, and Gilles Savard. Time-indexed formulations and the total weighted tardiness problem. *INFORMS Journal on Computing*, 2008.

Natashia Boland, Riley Clement, and Hamish Waterer. A bucket indexed formulation for nonpreemptive single machine scheduling problems. *INFORMS Journal on Computing*, 2016.

Herbert G Campbell, Richard A Dudek, and Milton L Smith. A heuristic algorithm for the n job, m machine sequencing problem. *Management science*, 16(10):B–630, 1970.

Andre A. Cire, Adam Diamant, Tallys Yunes, and Alejandro Carrasco. A Network-Based Formulation for Scheduling Clinical Rotations. *Production and Operations Management*, 28:1186–1205, 2019.

George B Dantzig and Philip Wolfe. Decomposition principle for linear programs. *Operations research*, 8(1):101–111, 1960.

Waraporn Fangrit, Hwa Jen Yap, Mukhtar Fatihu Hamza, Siow-Wee Chang, Keem Siah Yap, and Shen Yuong Wong. An efficiency improvement of flexible flow shop scheduling in automotive part company. *Intelligent Decision Technologies*, 14(4):493–506, 2020.

Amir Ghasemi, Radhia Azzouz, Georg Laipple, Kamil Erkan Kabak, and Cathal Heavey. Optimizing capacity allocation in semiconductor manufacturing photolithography area–case study: Robert bosch. *Journal of Manufacturing Systems*, 54:123–137, 2020.

Jan Gmys. Exactly solving hard permutation flowshop scheduling problems on peta-scale gpu-accelerated supercomputers. *INFORMS Journal on Computing*, 34(5):2502–2522, 2022.

Amit Kumar Gupta and Appa Iyer Sivakumar. Job shop scheduling techniques in semiconductor manufacturing. *The International Journal of Advanced Manufacturing Technology*, 27(11):1163–1169, 2006.

Maw Maw Htay, Guo Shunsheng, and Asa Romeo Asa. Quality information flow of welding process in auto manufacturing. *International Journal of Scientific and Technology Research*, 2(4):2277–8616, 2013.

Cheng-Ting Huang, Tsung-Jung Hsieh, and Bertrand MT Lin. Data-driven scheduling for the photolithography process in semiconductor manufacturing. *Journal of Industrial and Management Optimization*, 21(3):1946–1963, 2025.

Selmer Martin Johnson. Optimal two-and three-stage production schedules with setup times included. *Naval research logistics quarterly*, 1(1):61–68, 1954.

Hyun-Jung Kim and Jun-Ho Lee. Three-machine flow shop scheduling with overlapping waiting time constraints. *Computers & Operations Research*, 101:93–102, 2019.

Sebastian Knopp, Stéphane Dauzère-Pérès, and Claude Yugma. A batch-oblivious approach for complex job-shop scheduling problems. *European Journal of Operational Research*, 263(1):50–61, 2017.

Daniel Kowalczyk and Roel Leus. A Branch-and-Price Algorithm for Parallel Machine Scheduling Using ZDDs and Generic Branching. *INFORMS Journal on Computing*, 30:768–782, 2018.

Daniel Kowalczyk, Roel Leus, Christopher Hojny, and Stefan Røpke. A flow-based formulation for parallel machine scheduling using decision diagrams. *INFORMS Journal on Computing*, page ijoc.2022.0301, 2024.

D Phanindra Kshatra, S Akhil, U Kiran, and Y Vineeth. Process design and system layout for an automobile manufacturing and assembly plant. *Int. J. Innov. Technol. Explor. Eng*, 9:440–446, 2019.

Zhiyuan Li, Desheng Wu, Cheng He, Xiaofeng Gan, and Peichun Chen. Intelligent scheduling of automotive stamping workshops based on an enhanced genetic algorithm. *International Journal of Computers Communications & Control*, 20(1), 2025.

Wenqian Liu, Lindong Liu, and Xiangtong Qi. Drone resupply with multiple trucks and drones for on-time delivery along given truck routes. *European Journal of Operational Research*, 318:457–468, 2024.

Sreenath Chalil Madathil, Siddhartha Nambiar, Scott J Mason, and Mary E Kurz. On scheduling a

photolithography area containing cluster tools. *Computers & Industrial Engineering*, 121:177–188, 2018.

Bernardo Martin-Iradi, Dario Pacino, and Stefan Ropke. The multiport berth allocation problem with speed optimization: Exact methods and a cooperative game analysis. *Transportation Science*, 56(4): 972–999, 2022.

Rafael Morais, Teobaldo Bulhões, and Anand Subramanian. Exact and heuristic algorithms for minimizing the makespan on a single machine scheduling problem with sequence-dependent setup times and release dates. *European Journal of Operational Research*, 315:442–453, 2024.

Rafael Muñoz-Sánchez, Iris Martínez-Salazar, José Luis González-Velarde, and Yasmín Á Ríos Solís. Two hybrid flow shop scheduling lines with assembly stage and compatibility constraints. *Plos one*, 19(6):e0304119, 2024.

Muhammad Nawaz, E Emory Enscore Jr, and Inyong Ham. A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. *Omega*, 11(1):91–95, 1983.

Daniel Oliveira and Artur Pessoa. An improved branch-cut-and-price algorithm for parallel machine scheduling problems. *INFORMS Journal on Computing*, 32:90–100, 2020.

Yunpeng Pan and Leyuan Shi. On the equivalence of the max-min transportation lower bound and the time-indexed lower bound for single-machine scheduling problems. *Mathematical Programming*, 110: 543–559, 2007.

Kyungsu Park and James R Morrison. Controlled wafer release in clustered photolithography tools: Flexible flow line job release scheduling and an lmolp heuristic. *IEEE Transactions on Automation Science and Engineering*, 12(2):642–655, 2014.

Artur Pessoa, Eduardo Uchoa, Marcus Poggi de Aragão, and Rosiane Rodrigues. Exact algorithm over an arc-time-indexed formulation for parallel machine scheduling problems. *Mathematical Programming Computation*, 2:259–290, 2010.

M Pinedo. Theory, algorithms, and systems scheduling, fourth, 2012.

DY Sha, SY Hsu, ZH Che, and CH Chen. A dispatching rule for photolithography scheduling with an on-line rework strategy. *Computers & Industrial Engineering*, 50(3):233–247, 2006.

Dvir Shabtay and Enrique Gerstl. Coordinating scheduling and rejection decisions in a two-machine flow shop scheduling problem. *European Journal of Operational Research*, 316(3):887–898, 2024.

Francis Sourd. New Exact Algorithms for One-Machine Earliness-Tardiness Scheduling. *INFORMS Journal on Computing*, 21:167–175, 2009.

Ling-Huey Su. A hybrid two-stage flowshop with limited waiting time constraints. *Computers & Industrial Engineering*, 44(3):409–424, 2003.

Reha Uzsoy, Chung-Yee Lee, and Louis A Martin-Vega. A review of production planning and scheduling models in the semiconductor industry part i: system characteristics, performance evaluation and production planning. *IIE transactions*, 24(4):47–60, 1992.

J.M. Van Den Akker, C.P.M. Van Hoesel, and M.W.P. Savelsbergh. A polyhedral approach to single-machine scheduling problems. *Mathematical Programming*, 85:541–572, 1999.

J.M. Van Den Akker, C.A.J. Hurkens, and M.W.P. Savelsbergh. Time-indexed formulations for machine scheduling problems: Column generation. *INFORMS Journal on Computing*, 12:111–124, 2000.

Jihong Yan and Chenglong Wang. Stamping workshop scheduling optimization considering multiple constraints under time-of-use electricity pricing. *Computers & Industrial Engineering*, page 111199, 2025.

Dar-Li Yang and Maw-Sheng Chern. A two-machine flowshop sequencing problem with limited waiting time constraints. *Computers & Industrial Engineering*, 28(1):63–70, 1995.

Claude Yugma, Jakey Blue, Stéphane Dauzère-Pérès, and Ali Obeid. Integration of scheduling and advanced process control in semiconductor manufacturing: review and outlook. *Journal of Scheduling*, 18(2):195–205, 2015.

Peng Zhang, Youlong Lv, and Jie Zhang. An improved imperialist competitive algorithm based photolithography machines scheduling. *International Journal of Production Research*, 56(3):1017–1029, 2018.