OmniaScience

Journal of Industrial Engineering and Management
JIEM, 2026 – 19(1): 58-81 – Online ISSN: 2013-0953 – Print ISSN: 2013-8423
https://doi.org/10.3926/jiem.8710

A Batch Scheduling Model on Unrelated Parallel Machines with Resource Constraints and Sequence-Dependent Setup Time to Minimize Total Actual Flow Time

Rinto Yusriski[1*], Andri Rachmat Kumalasian Nasution[1,2], Mohammad Mi'radj Isnaini[2], Abdul Hakim Halim[2]

[1]*Universitas Jenderal Achmad Yani (Indonesia)*

[2]*Institut Teknologi Bandung (Indonesia)*

[*]*Corresponding author: rinto.yusriski@lecture.unjani.ac.id*
33421001@mahasiswa.itb.ac.id, isnaini@itb.ac.id, ahakimhalim@itb.ac.id

Received: January 2025
Accepted: November 2025

Abstract:

Purpose: This study develops a batch scheduling model on unrelated parallel machines with resource constraints and sequence-dependent setup times to minimize total actual flow time. The research is motivated by challenges in industrial settings where machines have different processing capabilities and setup times, impacting scheduling efficiency.

Design/methodology/approach: An algorithm is proposed using three steps: (1) sequencing jobs using the Longest Due Date (LDD) rule, (2) allocating job demands to machines based on capacity, and (3) solving the batch scheduling problem for each machine using a backward scheduling approach. Numerical experiments were conducted to evaluate the model against optimal solutions produced by enumeration algorithms.

Findings: The proposed algorithm achieves an average efficiency of 99.32% compared to the enumeration algorithm, with minimal deviation (0.4 %). The results validate key propositions for efficient scheduling, including prioritizing jobs by due dates, allocating demands based on machine capacity, and sequencing batches by size to minimize delays.

Research limitations/implications: The model assumes static machine conditions and predefined job parameters, limiting its application in static environments. Future research could extend this approach to incorporate real-time job arrivals or machine breakdown scenarios.

Practical implications: The algorithm offers a practical tool for industries to optimize batch scheduling on unrelated parallel machines, enhancing production efficiency and reducing operational costs.

Social implications: By improving production scheduling, the model indirectly supports sustainable manufacturing practices through optimized resource utilization.

Originality/value: This study provides a novel integration of backward scheduling with resource constraints and sequence-dependent setup times, addressing a gap in scheduling research on unrelated parallel machines.

Keywords: batch scheduling, unrelated parallel machine, total actual flow time, sequence-dependent setup, resource constraints

To cite this article:

Yusriski, R., Nasution, A.R.K., Isnaini, M.M., & Halim, A.H. (2026). A batch scheduling model on unrelated parallel machines with resource constraints and sequence-dependent setup time to minimize total actual flow time. *Journal of Industrial Engineering and Management*, 19(1), 58-81. https://doi.org/10.3926/jiem.8710

1. Introduction

Most industrial jobs require scheduling and limited resources to maximize the goal (Baker & Trietsch, 2009; Georgiadis, Elekidis & Georgiadis, 2019). One of the topics with the most significant amount of research is the parallel machine (PM) scheduling problem (Alhajjar & Mohammed, 2023; Lee & Jang, 2019; Li, Zhang, Leung & Yang, 2016; Mokotoff, Jimeno & Gutiérrez, 2001; Senthilkumar, Kannan & Madesh, 2017; Xu & Nagi, 2013). In this situation, several jobs must be processed on several machines simultaneously, and each machine handles each task independently, one at a time. There are three categories of parallel machine scheduling problems: identical parallel machines, uniform machines, and unrelated machines (Cheng & Sin, 1990; Muñoz-Villamizar, Santos, Montoya-Torres & Alvaréz, 2019). The identical machine category problems show that all machines have the same capability and speed for processing jobs. The uniform machine category problems assume that each machine has different capabilities and speeds for processing all jobs. Meanwhile, the unrelated machine category shows that each machine has different capabilities and speeds when processing each job.

This paper discusses the problem of batch scheduling on unrelated parallel machines with sequence-dependent setup time (BS-UPMRS) in the JIT environment. The motivation for the research is based on problems at the weaving machine workstation. Several jobs are scheduled to be processed in parallel on group warp machines. The warp machine creates grey (the in-process cloth) from weaving booms, which are then refilled with a new one, with setup times varying depending on the job and the replacement boom. If the replacement weaving boom comes from the same job, the setup time is the same length as the previous one, but it will be different if the replacement boom comes from another job. Job processing times vary based on process conditions and machine efficiency. The company aims to minimize flow time and maintain on-time delivery by returning weaving boom wheels simultaneously after processing. This research uses the total actual flow time as an objective to determine the sequence of jobs, calculate the number of machines for each job, and determine the batch scheduling solution in each machine.

2. Literature Review

The literature often overlooks the study of scheduling for unrelated parallel machine problems (UPM) (Kazemi, Mahdavi-Mazdeh, Rostami & Heydari, 2021; Muñoz-Díaz, Escudero-Santana & Lorenzo-Espejo, 2024; Olteanu, Sevaux & Ziaee, 2022). Kazemi et al. (2021) address the integration of production and distribution scheduling in a non-identical parallel machine environment using improved genetic algorithms (IGA). The objective is to minimize total job tardiness and delivery costs within a supply chain system by proposing a mixed-integer linear programming model and efficient metaheuristic approaches for solving large-scale problems. The study demonstrates that the Improved Genetic Algorithm (IGA) achieves an average deviation of up to 0.29% from the optimal solution for small-scale problems, compared to 0.47% for the standard Genetic Algorithm (GA). For large-scale problems, the Relative Percentage Deviation (RPD) averages 1.55% for IGA and 3.83% for GA, highlighting the superior performance of the proposed IGA in minimizing deviations while addressing complex scheduling scenarios. Olteanu et al. (2022) found that Simulated Annealing outperformed MILP and the Greedy Constructive Heuristic, achieving the best results with an average gap of less than 8% and producing near-optimal solutions in over 90% of cases for large-scale problems. While MILP performed well on smaller instances, it required extensive computation time, and the Greedy Heuristic, despite its speed, showed significantly higher gaps from the optimal solution, often exceeding 30%. Muñoz-Díaz et al. (2024) found that Tabu Search outperformed Simulated Annealing and the Constructive Heuristic, achieving the best results in over 96% of cases. While Simulated Annealing performed well on smaller problems, its consistency decreased as problem size grew.

Numerous publications have covered UPM setups in the past decade (Agárdi & Nehéz, 2021; Foroutan, Shafipour, Rezaeian & Khojasteh, 2024; Khanh-Van & Van-Hop, 2021). Agárdi and Nehéz (2021) proposed a genetic algorithm for an efficient solution. The authors found that setup time depends on the machine and job sequence, leading to decisions on assignment and sequence. Foroutan et al. (2024) discuss the scheduling of unrelated parallel machines with family setups and soft time windows, aiming to minimize weighted earliness and tardiness while maximizing the number of Just-in-Time jobs. The study compares four metaheuristic algorithms: Simulated Annealing, Artificial Immune System, Genetic Algorithm, and Ant Colony Optimization. The results demonstrate that the Ant Colony Optimization algorithm, combined with a repair strategy and local search, outperforms the others, achieving an average improvement in objective function values of 2.12% to 8.52% and a relative deviation of 10.35%. The research by Khanh-Van & Van-Hop (2021) develops the Genetic Algorithm with Initial Sequence based on the Earliness-Tardiness criterion on Parallel Machines (GAISETP) to address unrelated parallel machine (UPM) scheduling problems with sequence-dependent setup times and machine capacity constraints. The algorithm demonstrates superior performance over previous methods, achieving near-optimal solutions in a case study of automotive component manufacturing. Specifically, GAISETP achieves an average gap of less than 10% compared to optimal solutions for small-sized problems and approximately 6.5% for large-sized problems. The performance of the algorithm is evaluated using criteria such as makespan, total earliness, and total tardiness, combined into a single objective function.

Kong, Liu, Pei, Pardalos and Mladenovic (2020); Miao, Zhang and Cao (2011); and Shahvari & Logendra (2017) have explored batch scheduling on unrelated parallel machines (BS-UPM). Kong et al. (2020) explore parallel-batching scheduling with nonlinear processing times, focusing on single and unrelated parallel machines, introducing an optimal algorithm for single-machine settings and a hybrid SFLA-VNS meta-heuristic—combining Shuffle Frog Leap Algorithm (SFLA) and Variable Neighbourhood Search (VNS)—to effectively address NP-hard unrelated parallel machine problems. The paper aims to minimize the makespan by optimizing batch formation and scheduling while accounting for nonlinear job deterioration. Computational experiments demonstrate the superiority of SFLA-VNS, achieving better convergence rates and solutions than other algorithms, particularly for large-scale instances, with improvements of up to 3.8% in average objective values. Miao et al. (2011) explore bounded parallel-batch scheduling problems for deteriorating jobs on single and multiple machines, where processing time increases linearly with start time. The paper introduces an optimal algorithm for the single-machine scenario and an FPTAS for parallel machines with identical release dates, while proving NP-hardness for single machines with distinct release dates. It contributes algorithms for specific cases, emphasizing their computational complexity and potential extensions for future research. Shahvari and Logendran (2017) propose a model for batch scheduling on unrelated-parallel machines, aiming to minimize a bi-criteria objective function that combines total weighted completion time and total weighted tardiness. The research introduces an enhanced Tabu Search algorithm with three levels of search (central, outside, and inside) that iteratively refines batch compositions, batch sequencing, and job sequencing to address the NP-hard nature of the problem effectively. Experimental results show the proposed method achieves up to 37% improvement in objective function value compared to group scheduling, and computational efficiency is enhanced by up to 40% through theoretical lemmas to eliminate ineffective search neighbourhoods. Shahvari, Logendran and Tavana (2022) proposes an efficient model-based branch-and-price algorithm to address batching and scheduling problems on unrelated-parallel machines, focusing on minimizing a linear combination of total weighted completion time and total weighted tardiness. The model integrates a machine learning-based random forest algorithm for determining lower bounds on batch sizes and reformulates a mixed-integer linear programming model using flow conservation constraints to reduce computational complexity. The branch-and-price algorithm demonstrated superior performance, achieving optimal solutions with significant reductions in computational time compared to existing benchmarks, while maintaining high solution quality across various problem scales.

Researchers in BS-UPM have explored the JIT field (Goli & Keshavarz, 2022; Halim, Miyazaki & Ohta, 1991; Zarandi & Kayvanfar, 2015). Goli & Keshavarz (2022) examined a sequence-dependent group scheduling problem on parallel machines within a Just-In-Time (JIT) framework, aiming to minimize total weighted earliness and tardiness. The study begins by developing a mathematical model suitable for solving small-sized instances of the problem. Recognizing the problem's NP-hard nature, the researchers proposed two meta-heuristic algorithms to

find near-optimal solutions: The Biogeography-Based Optimization (BBO) algorithm, introduced as a novel approach, and the Variable Neighborhood Search (VNS) algorithm, a widely recognized method. To assess the effectiveness of the proposed model and algorithms, extensive computational experiments were conducted. The results demonstrated the efficiency of both algorithms in terms of speed and solution quality, with the BBO algorithm achieving a maximum gap of 1.04% and the VNS algorithm a slightly higher gap of 1.35%. These findings underscore the potential of BBO and VNS for addressing complex scheduling problems in JIT environments. Halim et al. (1991) proposed a single-job batch-scheduling algorithm for distributing parts and solving the batch-scheduling problem on parallel machines. The researchers use the total actual flow time as an objective, defined as the total time interval of all parts in all batches flowing in the shop from arrival to the due date. This objective has proven to minimize flow time and on-time delivery simultaneously (Kurniawan, Yusriski, Isnaini, Anas & Halim, 2021; Kurniawan, Yusriski, Isnaini, Ma'Ruf & Halim, 2024; Maulidya, Suprayogi, Wangsaputra & Halim, 2020; Yusriski, Sukoyo, Samadhi & Halim, 2015; 2016; 2018; Yusriski, Astuti, Ilham & Zahedi, 2019; Yusriski, Astuti, Biksono & Wardani, 2021). Zarandi & Kayvanfar (2015) investigated a bi-objective scheduling problem on identical parallel machines, integrating the Just-In-Time (JIT) philosophy to minimize the total costs of tardiness, earliness, job processing time adjustments, and makespan. The study introduced an innovative approach using the "bi-objective parallel net benefit compression-net benefit expansion" (BPNBC-NBE) heuristic, which allows for flexible compression or expansion of job processing times within defined limits. To solve this complex problem, the researchers applied two multi-objective meta-heuristic algorithms, Non-Dominated Sorting Genetic Algorithm II (NSGAII) and Non-Dominated Ranking Genetic Algorithm (NRGA). The findings revealed that NRGA excelled in achieving better convergence towards the Pareto-optimal front, while NSGAII demonstrated a wider spread across solutions. This research offers significant advancements in JIT scheduling by effectively balancing delivery precision with production efficiency, providing valuable insights for optimizing complex manufacturing systems and industrial processes.

3. Model and Solution Method

This section discusses the problem formulation, describes a mathematical model, and develops a solution method, including the proposed algorithm.

3.1. Problem Formulation

Multiple jobs with individual due dates are scheduled on a workstation containing unrelated machines. Each job consists of product units distributed to the machines, the so-called sub-jobs. Since each machine's capabilities differ, sub-job sizes can vary, even though they come from the same job. The company can manage the arrival of material to the shop at the right time and quantity (JIT environment), so to minimize inventory cost, each machine can process the sub-lot into some batches. The setup time is needed before any machine processes a batch; the length depends on the machine's capabilities. The objective is to minimize the total actual flow time, and the decision is to determine the sequence of jobs, the number of machines allocated to process each job, the number of batches, batch sizes, and the sequence of the resulting batches on each machine. Since the actual flow time is an objective, this research adopts the backward scheduling methods. The notation and mathematical model are as follows.

Index		
k	:	the job index $k = 1, \ldots, K$
m	:	the machine index $m = 1, \ldots, M$
i	:	the batch index $i = 1, \ldots, N_{km}$
Parameter		
n_k	:	the demands of job k
d_k	:	the due date of job k
Δ_{km}	:	the length between the due date of job k and the completion time of first batch scheduled on machine m by the backward scheduling approach
t_{km}	:	the processing time of job k on machine m
s_{km}	:	the setup time of the job k on machine m

Variable		
M	:	the number of machines (group machine) which is allocated to process a job
K	:	the number of jobs
A_{km}	:	sub-demand, the unit quantity of job k allocated to machine m
N_{km}	:	the number of batches of job on a machine m
$B_{km[i]}$	:	the starting time of the batch sequence i^{th} job k on machine m
$Q_{km[i]}$	:	the size of the batch sequence i^{th} job k on machine m
Objective		
F^a	:	the total actual flow time of all jobs

$$F^a = \sum_{k=1}^{K}\sum_{m=1}^{M}\left\{\sum_{i=1}^{N_{km}}\left(\sum_{j=1}^{i}\Delta_{[k]m[j]} + t_{[k]m}Q_{[k]m[j]} + s_{[k]m}\right) - s_{[k]m}\right\}Q_{[k]m[i]} \quad (1)$$

Subject to:

$$\sum_{m=1}^{M} A_{km} = n_k;\ k = 1, \ldots, K; \quad (2)$$

$$\sum_{i=1}^{N_{km}} Q_{kmi} = A_{km};\ k = 1, \ldots, K; \quad (3)$$

$$\Delta_{[k]m} + \sum_{i=1}^{N_{[k]m}}\left(t_{[k]m}Q_{[k]m[i]}\right) + s_{[k]m} \le d_{[k]}; i = 1, \ldots N_{km};\ k = 1, \ldots, K;\ m = 1, .., M; \quad (4)$$

$$B_{[k]m[1]} + t_{[k]m}Q_{[k]m[1]} + \Delta_{[k]m} = d_{[k]};\ k = 1, \ldots, K;\ m = 1, .., M; \quad (5)$$

$$\Delta_{[k]m} = max\left\{0, \sum_{i=1}^{N_{km}} t_{[k]m}Q_{[k-1],m[i]} + N_{[k]m}s_{[k]m} - \left(d_{[k]} - d_{[k+1]}\right)\right\};$$
$$\text{where } k = 1, \ldots, K; m = 1, .., M; \quad (6)$$

$$Q_{[k]mi} \ge 0, N_{[k]m} \ge 1\ and\ Integer; k = 1, \ldots, K;\ m = 1, .., M;\ i = 1, \ldots, N_{[k]m}; \quad (7)$$

Equation (1) is the objective function, minimizing the total actual flow time for multi-job unrelated parallel machines. Constraint (2) and (3) state the material balance. Constraint (2) shows that the total unit of a job allocated on all parallel machine is equal to that demand. Constraint (3) shows the total number of units in all batches of job scheduled on each machine must be equal to the total unit allocated to that machine. Constraints (4) state that all batches of jobs are processed on any parallel machines in the length of the scheduling period (between t = 0 and their individual due date). Constraint (5) states that the first batch of job j scheduled in the unrelated m parallel machine using a backward scheduling approach must be completed at their individual due date (d) exactly. Constraints (6) state the length between the due date of the job j and the completion time of first batch scheduled on machine m. Constraint (7) states that the minimum batch number and the minimum batch sizes are one, and the element is a natural number.

3.2. Problem Solution

This section will determine the decision variables along with the formula for calculating the minimum total actual flow time. The Six decisions are proposed: the job's sequencing, the demand allocation, the resource (machines) priority, the batch scheduling decisions consist of the decision of the number of batches, the batch sizes, and the scheduling of resulting batches.

3.2.1. Decision 1: Job's Sequencing

Multiple jobs are scheduled on parallel unrelated machines with an individual due date. Since this study adopts the backward scheduling approach, the jobs must be scheduled using the proposition as shown as follows.

Proposition 1. *Suppose there are K jobs with their respective due date scheduled on an unrelated parallel machine using the backward scheduling approach. In that case, the total actual flow time of jobs can be minimized by scheduling the jobs using the Longest Due Date (LDD) rule.*

Proof. *Suppose there are K jobs* $k = 1, \ldots, K$ *with an individual due date* $(d_1, \ldots, d_K)$ *respectively. The Jobs are scheduled using two schedule sequences, namely S and Z , adopting a backward scheduling approach. The S is a job sequence using the Longest Due Date (LDD) rule, while Z is a sequence using other rules so that the jobs sequence is* $S \neq Z$. *Let's assume there is a pair of jobs, namely a and b* $(a, b \in k)$ *with* $d_a \geq d_b$, *Position a precedes b in S, but vice versa in Z. If the positions of a and b are interchanged, then the value of* F^a *increases in S and decreases in Z. Applying the pairwise interchange method to all jobs so that Z = S produces an optimal schedule Z. This proves that the LDD rule produces an optimal solution to minimize the total actual flow time.* ■ **(proven).**

3.2.2. Decision 2: Demand Allocation

In the case of parallel machines, decisions on demand allocation are influenced by machines available capacity where each machine have a unique parameter, including job processing time and machine setup time. It leads to different machine capacities when the machine processes the jobs. The capacities of the machines are calculated using the following formula.

$$A_{[k]m} = \frac{n_{[k]m}+V}{t_{[k]m}W} - \frac{(\Delta_{km}+s_{[k]m})}{t_{[k]m}}; \; k = 1, \ldots, K; \; m = 1, \ldots, M$$

$$\text{where } V = \sum_{i=1}^{M}\left(\frac{\Delta_{[k]i}+s_{[k]i}}{t_{[k]i}}\right) \text{ and } W = \sum_{i=1}^{M}\left(\frac{1}{t_{[k]i}}\right); \; i \in m \tag{8}$$

3.2.3. Decision 3: Resource (Machines) Priority

The machine priority for allocating the demand jobs can be obtained by following the Proposition as follows.

Proposition 2. *If there are M unrelated parallel machines with their processing time and capacities will be assigned to process a job during a scheduled period, then selecting the machines priority for allocating the demand jobs obtained by the increasing of machine capacity,* $t_{[1]}C_{[1]} \leq \ldots \leq t_{[M]}C_{[M]}$.

Proof. *Let it see the Equation (1). Assume there is one job with one batch to be scheduled and so Equation (1) can be write as follows:* $F^a = \sum_{m=1}^{M} t_m Q_m^2$. *Partial differential* ∂F^a *over* ∂Q_m *found* $\frac{dF^a}{dQ_m} = \sum_{m=1}^{M} 2t_m Q_m$. *If the batch size on the machine equal to that capacity* $(Q_m = C_m)$ *so min* F^a *can be found when min* $2t_m C_m$ *since* $m = 1, \ldots, M$, *the priority allocation the demand to machine obtained by increasing of* $t_m C_m$, $t_{[1]}C_{[1]} \leq \ldots \leq t_{[M]}C_{[M]}$. ■ **(proven).**

Constrain (2) shows that the demand of jobs can be divided into several sub-demands in each machine. If the number of batches in each machine equal to 1, the batch sizes is a number of units that allocated to the machine, the formula for calculating the A_{km} with the continuous allocation size and integer size are as follows.

The continuous allocation calculated by formula:

$$A_{[k]m} = \frac{n_{[k]m}+V}{t_{[k]m}W} - \frac{(\Delta_{km}+s_{[k]m})}{t_{[k]m}}; \; k = 1, \ldots, K; \; m = 1, \ldots, M$$

$$\text{Where } V = \sum_{i=1}^{M}\left(\frac{\Delta_{[k]i}+s_{[k]i}}{t_{[k]i}}\right) \text{ and } W = \sum_{i=1}^{M}\left(\frac{1}{t_{[k]i}}\right); \; i \in m \tag{9}$$

The integer allocation calculated by formula:

$$A_{[k]m} = \begin{cases} \left[\frac{n_{[k]m}+V}{t_{[k]m}W} - \frac{(\Delta_{km}+s_{[k]m})}{t_{[k]m}}\right]; \; m = 1, \ldots, M-1; \\ n_k - \sum_{m=1}^{(M-1)} A_{[k]m}\,; \text{ otherwise;} \end{cases} \quad k = 1, \ldots, K; \; m = 1, \ldots, M \tag{10}$$

where $[X]$ is a round half up of X;

[k] is machine sequence which is determined by Proposition 2

How the solutions have been obtained is shown in the Appendix.

3.2.4. Decision 4: Number of Batches

This step discusses how to divide the sub-demand of the job in each machine into several batches (Q_{kmi}, $i = 1, \ldots, N_{km}$) and determine that sequence to minimize the total actual flow time. The decision method of this study adopted the solution of Halim, Miyazaki and Ohta, (1994a), which discussed batch scheduling model on a single machine common due date. The proposition of the sequence of the batch is as follows.

Proposition 3. *Suppose that there are N batches of job j scheduled on single machine with batch sizes (Q_{kmi}, $i = 1, \ldots, N_{km}$) respectively. The optimal backward sequence that minimizes F^a is obtained from arranging the batches in order of non-increasing batch sizes:*

Proof. *Looking at the formula as follows: $\sum_{i=1}^{N_{km}}(\Delta_{km} + t_{km}Q_{km[i]}) + s_{[j]m}$ in Equation (1). It can be observed that the value index of batches ($Q_{km[i]}$) are already in increasing order. Therefore, to minimize F^a, the batches should sequence in a non-increasing order of batch sizes (starting from the batch closest to the due date).* ■ **(proven).**

This research also uses the formula in Halim et al. (1994a) to calculate the number of batches (N_{km}, $N^0_{\max}$) and the batch sizes ($Q_{km[i]}$, $i = 1, \ldots, N_{km}$) with several adjustments regarding the problems discussed. The formula for calculating the number of batches is shown as follows.

$$N_{km} = \left(\frac{-Z}{s_{km}} + \frac{1}{s_{km}}\sqrt{Z^2 - 2A_{km}s_{km}t_{km}}\right) - 1$$

$$\text{where } Z = \frac{1}{2}A_{km}t_{km} + \left(\frac{A_{km}s_{km}}{\frac{L_k}{t_{km}} + \frac{s_{km}}{t_{km}} - A_{km}}\right) - \frac{1}{2}L_k - \frac{1}{2}s_{km} \tag{11}$$

$$L_k = d_{km} - s_{km} - \Delta_{km} - \max\{d_{(k+1)m}, 0\}$$

$$N^0_{max} = \frac{1}{2} + \sqrt{\frac{1}{4} + \frac{2A_{km}t_{km}}{s_{km}}} \tag{12}$$

3.2.5. Decision 5: Batch Sizes

The formula for calculating the batch sizes is shown as follows.

The continuous batch sizes calculated by formula:

$$Q_{km[i]} = \frac{A_{km}}{N_{km}} + \frac{s_{km}(N_{km}+1)}{2t_{km}} - \frac{s_{km}i}{t_{km}}; \; k = 1, \ldots, K; \; m = 1, \ldots, M \tag{13}$$

The integer allocation calculated by formula:

$$Q_{km[i]} = \begin{cases} max\left\{\left\lfloor \frac{A_{km}}{N_{km}} + \frac{s_{km}(N_{km}+1)}{2t_{km}} - \frac{s_{km}i}{t_{km}} \right\rfloor; 1\right\};\ i = 1, \ldots, (N_{km}-1); \\ A_{km} - \sum_{i=1}^{N_{km}-1} Q_{km[i]};\ i = N_{km} \end{cases} \quad (14)$$

$k = 1, \ldots, K;\ m = 1, \ldots, M,$

Where $\lfloor X \rfloor$ is a round down of X;

3.2.6. Decision 6: Scheduling of Resulting Batches

The scheduling of each batch can be determined by calculating its beginning time, which represents the arrival position of the batch on the production floor. The beginning time of each batch is derived by integrating Constraints (4), (5), and (6). Based on the combination of these three constraints, the following formula is obtained as follows.

$$B_{[k]m[i]} = d_{[k]} - \Delta_{[k]m} - \sum_{j=1}^{i} \left(t_{[k]m} Q_{[k]m[i]}\right) + (i-1)s_{[k]m}; \quad (15)$$

Where $i = 1, \ldots N_{km};\ k = 1, \ldots, K;\ m = 1, .., M;$

How the solutions have been obtained is shown in Halim, Miyazaki and Ohta (1994b).

4. Results and Discussion

This section discussed the proposed algorithm and examines the outcomes of the proposed algorithm when used in basic scenarios and numerical experiments. The purpose of using the suggested method on basic scenarios is to give a general idea of how the algorithm functions and the outcomes achieved. Meanwhile, the goal of numerical experiments is to obtain an understanding of algorithmic techniques for solving issues.

4.1. Result

Base on the solution in the last section, the proposed algorithm solution has developed as follows.

The Proposed Algorithm (PA)

Step 1: set parameters

Step 2: sequence the jobs [k] using the LDD rule obtained by Proposition 1; go to Step 3.

Step 3: start from k = [1] and go to step 4.

Step 4: determine the demand allocation

Step 4.1: calculate the machine capacities using Equation (8); Continue to Step 4.2.

Step 4.2: sequence the machine priority by Proposition 2; Continue to Step 4.3.

Step 4.3: determine the demand allocation using Equation (9) for continue batch sizes case and Equation (10) for integer batch sizes case; Go to Step 5.

Step 5: determine the number of batches, and the batch sizes, then schedule the resulting batches for each machine using the common due date algorithm by Halim et al. (1994b).

Step 5.1: start from m = 1, go to Step 5.2

Step 5.2: calculate $L_{[k]} = d_{[k]m} - s_{[k]m} - \Delta_{[k]m} - \max\{d_{[k+1]m}, 0\}$; go to step 5.3

Step 5.3: if $s_{km} + A_{km}t_{km} + \Delta_{km} \geq L_k$ Set $N_{km} = 1$ then go to Step 6; otherwise, go to Step 5.4.

Step 5.4: compute $N^0_{\max}$ using Equation (12) and go to Step 5.5.

Step 5.5: Set $N_{\max} = \lfloor N^0_{\max} \rfloor$ means the maximum integer less than or equal to $N^0_{\max}$

Go to Step 5.6.

Step 5.6: compute N^0_{km} using Equation (11)

Step 5.6.1:if $N^0_{km} \leq 1$ then set $N_{km} = 1$

Step 5.6.2:if $N^0_{km} \geq 1$ $N_{\max}$ then set $N_{km} = N_{\max}$

Step 5.6.3: if $1 < N^0_{km} < N_{\max}$ then set $N_u^0 = \lceil N^0_{km} \rceil$ where $\lceil N^0_{km} \rceil$ is the minimum integer greater than or equal to N^0_{km}.

if $(N_u^0 - 1)s_{km} + A_{km}t_{km} \leq L_k$, set $N_{km} = N_u^0$

otherwise $N_u^0 - 1$,

Continue to Step 5.7.

Step 5.7: determine the batch sizes. If the problem considers Continuous batch sizes, proceed to Step.5.7.1, otherwise go to Step 5.7.2.

Step 5.7.1: compute $Q_{km[i]}$ ($i = 1, \ldots, N$) using Equation (13)

Step 5.7.2: compute $Q_{km[i]}$ ($i = 1, \ldots, N$) using Equation (14)

Continue to Step 5.8.

Step5.8: sequence the batches $Q_{km[i]}$ in non-increasing $Q_{km[i]}$ in the backward approach, The largest $Q_{km[i]}$ scheduled close to the due date.

Continue to Step 5.9.

Step 5.9: calculate beginning time with Equation (15). Continue to Step 5.10.

Step 5.10: if $m < M$ set $m + 1$ and return to Step 5.2 otherwise go to Step 6.

Step 6: if $k < K$, set $k = [k + 1]$, then return to Step 4; otherwise, continue to Step 7.

Step 7: Calculate the total actual flow time using Equation (1) and **STOP.** ■

The application of the proposed algorithm can be seen in a simple case example as follows: There are two jobs ($k = 1,2$) scheduled to process on three unrelated machines to minimize total actual flow time. The parameters are shown in Table 1.

Job number, k	processing time, t_m (hours/ unit)			Setup time, s_m (hours/ unit)			Due date, d (hours)	Demand, n (unit)
	$m = 1$	$m = 2$	$m = 3$	$m = 1$	$m = 2$	$m = 3$		
1	1	1	2	2	1	1	10	13
2	2	1	1.5	3	4	2	20	15

Table 1. The parameters of the case

The continue batch size case completion. The application of the proposed algorithm for the continue batch size case yields the solution as follows. The sequence of jobs resulting from STEP 2 is 2-1 since the algorithm adopts the LDD rule. STEP 3 schedule job number 2. The result of STEP 4 is that the demand equal =15 units are distributed to the machines. The demand allocation for each consecutive machine ($m = 1, 2, 3$) is 3.538 units, 6.077 units, and 5.385 units. STEP 5 continues to determine the number of batches and batch sizes on all machines. For machine number 1 ($m = 1$), the calculation result shows the number of batches (N_{21}) is 1, and the batch size ($Q_{21[1]}$) is 3.538 units. It leads to the starts processing of batch on machine number 1 ($B_{21[1]}$) at 12.923th hours. For $m = 2$, $N = 1$, $Q_{22[1]}$ =6.077 units, and $B_{22[1]}$ at 13.923th hours. Meanwhile for $m = 3$, the solution found with $N = 1$, with the batch sizes are $Q_{23[1]}$ = 5.385 units. The start processing $B_{23[1]}$ at 11.923th hours. Since another job has not been scheduled, STEP 6 continues to the next job (job number 1). The solution is completed with the same steps as the previous job. The last step (STEP 7) calculated the total actual flow time and found 164.192 hours. The complete solution to this problem can be seen in Table 2.

k	n	m	Δ_{km}	A_{km}	N_{km}	$Q_{km[i]}$ $(i = 1, \ldots, N_{km})$	$B_{km[i]}$ $(i = 1, \ldots, N_{km})$	F_m^a	F^a
2	15	1	0	3.538	1	(3.538)	(12.923)	105.462	164.192
		2	0	6.077	1	(6.077)	(13.923)		
		3	0	5.385	1	(5.385)	(11.923)		
1	13	1	0.077	4.6	2	(3.3), (1.3)	(6.623), (3.323)	58.73	
		2	0.077	5.6	3	(2.867), (1.867), (0.867)	(7.056), (4.19). (2.323)		
		3	0.077	2.8	2	(1.433), (0.933), (0.433)	(7.056), (4.19). (2.323)		

Table 2. Complete solution for the case

The integer batch size case completion. The application of the proposed algorithm yields the solution as follows. The sequence of jobs resulting from STEP 2 is 2-1 since the algorithm adopts the LDD rule. Start from $k = 2$. The result of STEP 3 is that the capacities of the machine (C_{2m}, $m = 1, 2, 3$) are 3 units, 6 units, and 5 units consecutively, and the machine priority sequence is 1-3-2 resulting by STEP 4. The next step (STEP 5) is the demand allocation of each job appropriate with job sequence priority. The demand equal =15 units are distributed to the machines. Sub-Algorithm 1 is used to solve that. The demand allocation for each consecutive machine ($m = 1, 2, 3$) is 3 units, 7 units, and 5 units. The result of STEP 5 shows that the demand allocation is bigger than the capacities, which leads the completion time of the next job to be earlier than its due date. STEP 6 continues to determine the number of batches and batch sizes on all machines. The result of Sub Algorithm 2 is as follows. For machine number 1 ($m = 1$), the calculation result shows the number of batches (N_{21}) is 1, and the batch size ($Q_{21[1]}$) is 3 units. It leads to the starts processing of batch on machine number 1 ($B_{21[1]}$) at 14th hours. For $m = 2$, $N = 7$, $Q_{22[1]} = 7$ units, and $B_{22[1]}$ at 13th hours. Meanwhile for $m = 3$, the solution found with $N = 1$, with the batch sizes are $Q_{23[1]} = 5$ units. The start processing $B_{23[1]}$ at 12.5th hours. Since another job has not been scheduled, STEP 7 continues to the next job ($k = 1$). The solution is completed with the same steps as the previous job. The last step (STEP 8) calculated the total actual flow time and found 168.5 hours. The complete solution to this problem can be seen in Table 3.

j	n	m	Δ_{km}	C_{km}	A_{km}	N_{km}	$Q_{km[i]}$ $(i = 1, \ldots, N_{km})$	$B_{km[i]}$ $(i = 1, \ldots, N_{km})$	F^a
2	15	1	0	3	3	1	(3)	(14)	168.5
		2	0	6	7	1	(7)	(13)	
		3	0	5	5	1	(5)	(12.5)	
1	13	1	0	8	5	2	(4), (1)	(6), (3)	
		2	1	8	5	2	(3), (2)	(6), (3)	
		3	0	4	3	2	(2), (1)	(6), (3)	

Table 3. Complete solution for the case

An illustration of the solution schedule is shown in the Figure 1.

Figure 1 shows three parallel machines processing two jobs ($k = 1,2$) with due dates of 20 and 10 hours and demands of 15 and 13 units, respectively. Based on the calculations of the proposed algorithm, the scheduling sequence prioritizes the second job, followed by the first job. The second job, scheduled first, is represented in red and is distributed across the three machines as follows: machine 1 processes 3 units (in one batch), machine 2 processes 7 units (in one batch), and machine 3 processes 5 units (in one batch). The first job, scheduled second, is represented in blue and is distributed across the three machines as follows: machine 1 processes 4 and 1 units (divided into two batches), machine 2 processes 3 and 2 units (divided into two batches), and machine 3 processes 2

and 1 units (divided into two batches). The figure illustrates that the two jobs are scheduled in a backward approach, ensuring that completion times do not exceed the respective due dates.

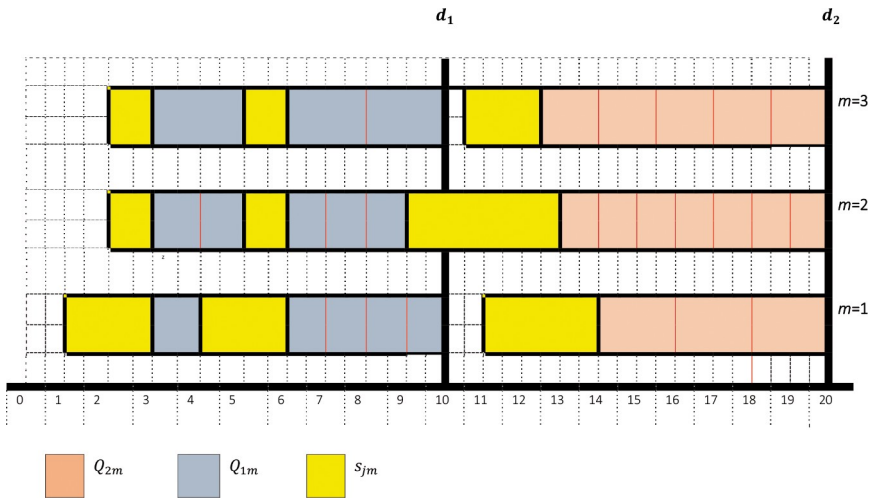

Figure 1. Gantt chart solution

4.2. Discussion

In this section, a comparative analysis is conducted between the performance of the proposed algorithm (PA) and the optimal solution outlined by the enumeration algorithm (EA). The aim is to assess how the proposed algorithm performs compared to the enumeration algorithm. The enumeration algorithm (EA) is obtained through the following steps:

4.2.1. The Enumeration Algorithm (EA)

Step 1: set input parameters

Step 2: sequence the jobs by Proposition 1, and go to Step 3.

Step 3: for each job resulting from Step 2:

Step 3.1: generate the alternative combination of demand allocation ($A_{[k]m}$, m = 1, …, M) using restricted integer partition with maximum value (RICMV) method, continue to Step 3.2.

Step 3.2: distribute each $A_{[k]m}$ to the machine by the Proposition 2, continue to Step 3.3

Step 3.3: for each $A_{[k][m]}$, generate the alternative combination of the batch sizes on each machine ($Q_{km[i]}$, i = 1, …, N) using the integer partition method. Go to Step 4.

Step 4: for each machine,

Step 4.1: combinate each alternative $Q_{km[i]}$ of a job with another job, continue to Step 4.2.

Step 4.2: calculate the total actual flow time of each combination (F_m^a) using Equation 1, continue to Step 4.3

Step 4.3: find the minimum total actual flow time for each machine; go to Step 5

Step 5: calculate the total actual flow time of all machines. **STOP.** ■

The completion of the last problem using the enumeration algorithm is as follows. Step 2 results in the sequence of jobs being 2-1. Step 3 produces the alternatives of the combination of demand allocation A_{km}. Here is an example for k = 2. The parameters of the RICMV method are the number of demands (n), the number of machines (M = 3), and the restricted of maximum value (r). the formula of r is calculated as follows.

$$r = \max\{C_{[k]m}\} + 1\,;\; k = 1, \dots, K;\; m = 1, \dots, M \tag{16}$$

The alternative solutions (A_{km}, m = 1, 2, 3) are shown in Table 4 as follows.

n	M	r	The RICMV combination solution without Proposition 2 (Step 3.1)	The RICMV combination solution (A_{km}) considering Proposition 2 (Step 3.2)
15	3	7	(7, 7, 1)	(1, 7, 7)
			(7, 6, 1)	(1, 7, 6)
			(7, 5, 3)	(3, 7, 5)
			(7, 4, 4)	(4, 7, 4)
			(6, 6, 3)	(3, 6, 6)
			(6, 4, 5)	(4, 6, 5)

Table 4. Solutions for demand distribution using the RICMV method

The solution of A_{km} is then used as a basis for enumerating decisions of N_{km} and $Q_{km[i]}$ using the integer partition method. One of the combinations (A_{km}, m = 1, 2, 3) in Table 4 is 3, 7, 5. The following is the solution for m = 3 (A_{km} = 5) which is solved with the integer partition method, is shown in Table 5 as follows.

A_{2m}, m = 1, 2, 3	A_{23}	N_2	$Q_{23[i]}$, i = 1, …, N_2
3, 7, 5	5	1	(5)
		2	(4, 1)
			(3, 2)
		3	(3, 1, 1)
			(2, 2, 1)
		4	(2, 1, 1, 1)
		3	(1, 1, 1, 1, 1)

Table 5. Solutions of demand distribution using the RICMV method (Step 3.3)

Step 4 is combining the $Q_{km[i]}$ for k = 1 and k = 2, continue with Step 5 to calculate the total actual flow time. The resulting example for this stage is shown in Table 6, which combines $Q_{23[1]}$ = 5 unit (k = 2, m = 3 and N_k = 1) with $Q_{13[i]}$ (k = 1, m = 3 and N_k = 1, …, 3).

$Q_{23[1]}$, N_2 = 1	Q_{13}	N_1	$Q_{13[1]}$, i = 1, …, N_1	Combination {($Q_{23[1]}$), ($Q_{23[i]}$)}	F_m^a	Min F_m^a
(5)	3	1	(3)	(5), (3)	55.5	
		2	(2, 1)	(5), (2, 1)	52.5	minimum
		3	(1, 1, 1)	(5), (1, 1, 1)	52.5	minimum

Table 6. Alternative solutions combination example (Step 4)

Table 6 shows the minimum F^a = 52.5 found from combination {(5), (2,1)} and {(5), (1,1,1)}. The next step, Step 5, is the final solution, shown in Table 7.

m	Final Combination	F_m^a	F^a
1	{(3), (4,1)}	41	168.5
2	{(7), (3,2)}	75	
3	{(5), (2,1)} or {(5), (1,1,1)}	52.5	

Table 7. Final solutions (Step 5)

Table 7 shows that one of the enumeration procedure solution provides the same solution as the proposed algorithm. The optimal combination of the enumeration procedure is founded with combination as shown in Tabel 8 as follows.

m	Optimal Combination	F_m^a	F^a
1	{(4), (3,2)}	60	167.5
2	{(6), (3,2,1)}	65	
3	{(3, 2), (1,1)}	42.5	

Table 8. Optimal solutions resulted by Enumeration Algorithm

As shown in Table 8, the enumeration solution provides a better result than the proposed algorithm. The effectiveness of the proposed algorithm is approximately 99.40% relative to the optimal solution.

The next step is to compare the two algorithms with several cases. The goal is to gain numerical experience regarding the performance of the proposed algorithm when compared with the optimal algorithm as measured by minimizing the total actual flow time (F^a) and computing the computer processing unit's (CPU's) time. The two algorithms are executed using the Visual Studio Community 2022 application with .NET 7 as a framework and C#11 as a programming language. The outcomes are reported using an Intel Xeon E5 v3, 6 cores, 20 GB of RAM, and an AMD Radeon R750 GPU. The result is shown in Table 9.

Number of windows	Number of machines	Number of testing	Proposed Algorithm		Enumeration Algorithm		% efficiency PA-EA	
			F^a	CPU's time (sec)	F^a	CPU's time (sec)	F^a	CPU's time (sec)
2	2	100	44.51	0.04	43.55	0.68	98.09 %	95.50 %
2	3	100	58.86	0.01	57.91	18.42	98.68 %	99.34 %
2	4	100	59.68	0.02	58.93	69.71	98.82 %	98.97 %
2	5	100	61.02	0.02	59.61	73.59	97.80 %	99.68 %
2	6	100	64.66	0.05	63.59	247.45	98.33 %	99.27 %
2	7	100	67.50	0.10	66.28	217.63	98.27 %	98.86 %
2	8	100	71.28	0.15	69.98	295.64	98.34 %	98.31 %
2	9	100	77.86	0.19	76.00	306.72	97.69 %	99.30 %
2	10	100	83.59	0.18	81.71	320.73	97.88 %	99.37 %
2	11	100	85.10	0.28	83.78	372.46	98.57 %	98.65 %
						Min	97.69 %	95.50 %
						Max	98.82 %	99.68 %
						Mean	98.25 %	98.73 %
						Standard Deviation	0.38 %	1.20 %

Table 9. Comparison results between the two algorithms

The experiment has been carried out more than 3,000 times, but in Table 9, only a sample of the results of 1,000 experiments is presented, considering that the results exhibit similar characteristics. Table 9 shows that the average percentage of efficiency F^a produced by the PA algorithm reached 98.25% when compared to the optimal solution produced by the EA algorithm, with a deviation of 0.38%. This indicates that the proposed algorithm consistently generates high-quality solutions very close to the optimum when compared with the enumeration algorithm. The small percentage difference mainly results from rounding operations in the model, specifically in Equation (10) for integer demand allocation and Equation (14) for integer batch size calculation. These two equations are interrelated: Equation (10) determines how the total job demand is discretely allocated to each machine, while Equation (14) computes the integer batch sizes based on that allocation. Rounding up or down at either stage may slightly alter the solution from the mathematical optimum, resulting in minor deviations in the final total actual flow time.

Table 9 also shows that the CPU time of the PA algorithm is significantly more efficient than that of the EA algorithm. The average CPU efficiency reached 98.73% with a deviation of 1.20%, confirming that the proposed algorithm maintains high computational performance even as the number of machines increases. For instance, when the number of machines increased to eleven, the PA algorithm required only 0.2845 seconds, compared to 372.46 seconds for the EA, resulting in a speed-up of over 1,300 times. These numerical results confirm that the proposed algorithm is suitable for solving complex batch scheduling problems efficiently. Furthermore, the experimental findings empirically validate the three propositions proposed in this study: (1) jobs should be scheduled using the Longest Due Date (LDD) priority rule, (2) job demands should be distributed according to machine capacity—where larger capacities receive larger demand allocations, and (3) batches should be sequenced such that larger batch sizes are scheduled closer to their respective due dates.

4.2.2. Branch-and-Bound (BNB) Algorithm

Although the Proposed Algorithm (PA) demonstrates high computational efficiency and consistently produces near-optimal solutions, it cannot guarantee global optimality due to its heuristic nature. In contrast, the Enumeration Algorithm (EA) ensures the exact optimal solution by exhaustively exploring all possible combinations; however, its computational effort increases exponentially with the problem size, making it impractical for large instances. To bridge the gap between efficiency and optimality, this study develops a Branch-and-Bound (BNB) algorithm. The BNB approach systematically explores feasible nodes while eliminating dominated or infeasible ones using the Lower Bound (LB) and Upper Bound (UB) concepts.

The Upper Bound (UB) represents the best total actual flow time (F^a) obtained thus far during the BNB exploration. Initially, the UB is set to the F^a value generated by the Proposed Algorithm (PA), which serves as an efficient near-optimal reference. Each time the BNB algorithm discovers a feasible schedule whose $LB_{[k]}$ is smaller than the current UB, the UB is updated accordingly. Mathematically, for each feasible solution as follows.

$$UB = min\left(UB, total\ LB_{[k]}\right);\ k = 1, \dots, K \tag{17}$$

Where $LB_{[k]}$ denotes the total actual flow time of all feasible job, starting from current job position until end position.

This definition ensures that the BNB search process always retains the best (minimum) total flow time discovered so far.

In contrast, the Lower Bound (LB) represents the theoretical minimum total actual flow time that could still be achieved by a partial or incomplete schedule under ideal conditions.

At each branching node, the LB serves as a decision criterion to determine whether a node is worth further exploration. If the calculated LB value of a node exceeds or equals the current UB, that node can be safely eliminated without affecting the global optimality of the final solution.

For a multi-job, multi-machine system, the LB of each job depends on its scheduling status.

The BNB algorithm adopts a hybrid LB formulation that combines discrete and continuous relaxation components:

1. For the active job (the job currently being scheduled), the discrete LB is computed based on its actual allocation to each machine:

$$LB_{[k]}^{dic} = \sum_{m=1}^{M} t_{[k],m} A_{[k],m};\ k = 1,..,K; m = 1,...,M \tag{18}$$

Where $t_{[k],m}$ is the processing time per unit job $[k]^{th}$ sequence on machine m

$A_{[k],m}$ is the number of allocated units of job $[k]^{th}$ sequence on machine m

For the unscheduled (future) jobs, a continuous relaxation is employed to represent the minimum attainable flow time assuming zero setup times and ideal machine utilization:

$$LB_{k}^{cont} = n_k^2 / \sum_{m=1}^{M} \left(1/t_{k,m}\right);\ k = 1,..,K;\ m = 1,...,M \tag{19}$$

Where n_k is the total number of units of job k.

Accordingly, the total LB at the current ($[k]^{th}$) position is formulated as follows:

$$total\ LB_{[k]} = \sum_{j=1}^{[k-1]} F_{[j]}^{a} + LB_{[k]}^{disc} + \sum_{g=[k+1]}^{K} LB_{g}^{cont};\ k = 1,..,K \tag{20}$$

If the computed $LB_p \geq UB$, the corresponding node is eliminated because it cannot produce a better solution than the current best. This condition forms the basis of the dominance pruning rule, which will be described as follows.

1. Prune-A (Out-of-Horizon) – A node is eliminated when backward scheduling yields a negative start time ($B_{[k],m} < max(d_{[k+1]}, 0)$), meaning that the process or setup of a batch would begin before next due date of time zero, making the schedule infeasible.
2. Prune-B (Dominance) – A node is eliminated when the total lower bound of the current node is not better than the incumbent upper bound ($UB = min(UB, total\ LB_{[k]})$).

The pruning mechanism described above serves as the logical foundation for constructing the complete Branch-and-Bound (BNB) algorithm. By combining the previously defined bounds (LB and UB) with the pruning rules, the BNB algorithm systematically explores feasible scheduling nodes while efficiently discarding infeasible or dominated ones. The algorithm consists of a structured sequence of decisions that correspond to the hierarchical nature of the batch scheduling problem, where each decision stage progressively refines the search space toward the global optimum.

Step 1: Initialization

Set all input parameters (t_{km}, s_{km}, d_k, n_k) for all jobs and machines.

Initialize $UB = F^a$ from the Proposed Algorithm (PA).

Step 2: Job Sequencing

Sequence jobs by Proposition 1 (LDD rule), obtaining ordered jobs $[k] = [1], \ldots, [K]$

Step 3: For each job k = [1] to [K] do

Step 3.1: Compute machine capacities using Equation (8):

Step 3.2: Sort machines in ascending order of C_m (Proposition 2).

Step 3.3: Generate all alternative demand allocations ($A_{[k]m}$)

using the Restricted Integer Combination with Maximum Value (RICMV) method. Let G be the total number of allocation alternatives.

Step 3.4: For each allocation g = 1 to G do

Step 3.4.1: Assign $A_{[k]m}$ to machines according to Proposition 2.

Step 3.4.2: For each feasible allocation, perform branching as follows:

Step 3.4.2.1: For each feasible number of batches $N_{[k]m} = 1$ to N_{max}:

Step 3.4.2.1.1: Generate all non-increasing batch partitions $Q_{[k]m[l]}$ using the Cartesian Product method.

Step 3.4.2.1.2: Schedule batches backward using Proposition 3 and calculate $B_{[k]m[l]}$ using Equation (15):

Step 3.4.2.1.3: Check horizon feasibility:

If $B_{[k],m} < max(d_{[k+1]}, 0)$ then eliminate node by Prune-A rule and continue.

Step 3.4.2.1.4: Compute total actual flow time of [k]: $F^a_{[k]} = \sum_{m=1}^{M} F^a_{[k]m}$ and continue

Step 3.4.2.1.5: Compute total LB using Equations 18-20

Step 3.4.2.1.6: If *total LB* ≥ *UB* then eliminate Prune-B rule; otherwise record the corresponding schedule as the incumbent.

Step 4: Termination

When all nodes are fathomed, report UB and its associated schedule as the global optimal solution. ■

Algorithm Evaluation and Example Results is as follows. The same two-job, three-machine example used for evaluating the PA and EA algorithms is also applied here to demonstrate the implementation of the proposed Branch-and-Bound (BNB) algorithm under backward scheduling. All job parameters, including processing times, setup times, due dates, and demands, remain identical to those defined previously.

The algorithm is executed sequentially, starting with Step 1 (Initialization), where all model parameters are defined and the upper bound (UB) is initialized using the Proposed Algorithm (PA), and Step 2 (Job Sequencing), which applies the Largest Due-Date (LDD) rule to fix the processing order as Job-2 followed by Job-1. This sequence provides the conditional search path for subsequent BNB exploration. The BNB procedure then follows the hierarchical decision flow (Steps 3.1–3.6) and employs two pruning rules: Prune-A (out-of-horizon) and Prune-B (dominance), using the lower-bound formulations as defined in the model.

Start from Job 2 (k = [1]) the result of step 3 is as follows. In Step 3.1, machine capacities are calculated using Equation (8), resulting in capacity are (4,7,5) which represents the maximum feasible allocation of Job-2 units to Machines 1–3. In Step 3.2, machines are sorted according to Proposition 2, which ensures that the RICMV allocations follow the same capacity-based priority. Based on this order, three feasible allocation alternatives are generated: (4, 7, 4), (4, 6, 5), and (3, 7, 5). For each allocation, the algorithm starts with a single-batch configuration N = (1, 1, 1) and incrementally increases the number of batches in Cartesian order following the established machines priority-Stepwise computation. The result of Step 3.3 until 3.4 for Job 2 is shown in Table 10 as follows.

Allocation	Batch decision (N; Q)	$F_{[1]}{}^a$	Total LB	Decision
(4, 7, 4)	N = (1, 1, 1); Q = ([4], [7], [4])	103.5	169.5	Prune-B
(3, 7, 5)	N = (1, 1, 1); Q = ([3], [7], [5])	101.8	167.8	Prune-B
(4, 6, 5)	N = (1, 1, 2); Q = ([4], [6], [3, 2])	100.5	166.5 < UB_0	Feasible – keep as incumbent
(4, 6, 5)	(1, 2, 1), (2, 1, 1), (1, 1, 3) …	101.9 – 102.8	167.8 – 168.8	Prune-B

Table 10. The Result of Step 3 for Job-2

As shown in Table 10, Job-2 obtains the optimal allocation on machines m1, m2, and m3, with processing quantities of (4, 6, and 5), respectively. The corresponding number of batches (N) for each machine is (1, 1, and 2),

with batch sizes of ([4], [6], and [3, 2]). The backward scheduling is feasible under the horizon constraint, and the objective function value (f) is 100.5.

The stepwise computation then continues for Job-1 (k = [2]) as described below. Given $F_{[1]}{}^{a}$ = 100.5, the subsequent job is evaluated using the same branching logic. Six allocation candidates are examined: (6,4,3), (4,6,3) (5,5,3), (7,3,3), (5,4,4), and (5,6,2). Because Job-1 is the last job, its LB corresponds directly to its actual flow-time contribution ($F_{[2]}{}^{a}$). The summary result of Step 3 for Job 1 is shown in Table 11 as follows.

Allocation	Batch decision (N; Q)	$F_{[2]}{}^{a}$	Total LB = 100.5 + LB	Decision
(6,4,3)	N = (1, 1, 1); Q = ([6], [4], [3])	71.0	171.5	Prune-B
(4,6,3)	N = (1, 1, 1); Q = ([4], [6], [3])	69.2	169.7	Prune-B
(5,5,3)	N = (1, 1, 1); Q = ([5], [5], [3])	69.8	170.3	Prune-B
(7,3,3)	N = (1, 1, 1); Q = ([7], [3], [3])	72.0	172.5	Prune-B
(5,4,4)	N = (1, 1, 1); Q = ([5], [4], [4])	70.1	170.6	Prune-B
(5,6,2)	N = (2, 3, 2); Q = ([3, 2], [3, 2, 1], [1, 1])	67.0	167.5 = UB	Feasible – Incumbent final

Table 11. The Result of Step 3 for Job-1

Based on Table 11, it can be seen that after the (5,6,2) configuration is obtained, the total F^{a} = 100.5 + 67.0 = 167.5, which tightens the upper bound (UB) and prunes all remaining nodes. Both jobs satisfy the backward-horizon feasibility without triggering *Prune-A*, while *Prune-B* effectively eliminates all dominated nodes once UB reaches 167.5. Finally, in Step 4, the algorithm reports UB = 167.5 as the optimal solution.

To further evaluate the computational efficiency and structural behavior of the optimization methods, a comparative experiment was conducted between the Branch and Bound (BNB) algorithm and the Enumeration Algorithm (EA) under identical scheduling environments. While both methods guarantee global optimality, their internal search mechanisms differ significantly: the EA performs exhaustive enumeration of all feasible combinations, whereas the BNB approach employs pruning strategies to systematically eliminate dominated or infeasible nodes based on bounding criteria. The performance comparison focuses on three primary indicators, total actual flow time, CPU execution time, and the resulting performance index, to assess how pruning influences computational time without compromising optimality. The summary of this comparison is presented in Table 12 as follows.

Based on Table 12, it can be observed that the Branch and Bound (BNB) algorithm consistently achieves the same optimal total actual flow time (F^{a}) as the Enumeration Algorithm (EA) across all test scenarios, with an average efficiency of 100% and zero deviation in solution accuracy. This confirms that both methods are capable of attaining globally optimal results under identical scheduling configurations. However, substantial differences are evident in the computational time (CPU) required to reach those solutions. The BNB method demonstrates an average CPU efficiency of 88.53% with a standard deviation of 1.68%, indicating that it performs significantly faster than the exhaustive enumeration procedure. In particular, the CPU performance index—which represents the relative reduction in computational effort—averages 90.65%, signifying that the BNB algorithm effectively reduces computation time by more than nine times on average while preserving the same level of optimality as the EA.

The observed performance improvement can be directly attributed to the pruning mechanism inherent in the Branch and Bound structure. While the Enumeration Algorithm (EA) explores every possible combination of job allocations and batch sequences exhaustively, the BNB method systematically eliminates non-promising or dominated nodes during the search process. This is achieved through the application of Lower Bound (LB) and Upper Bound (UB) thresholds, along with two pruning rules—Prune-A (Out-of-Horizon) and Prune-B (Dominance)—which discard infeasible or suboptimal branches early in the computation. As a result, the BNB algorithm focuses only on promising solution regions without compromising global optimality. This selective search strategy explains the substantial reduction in CPU time while maintaining perfect efficiency in F^{a}. Therefore, the

BNB serves as a practical and computationally efficient alternative to exhaustive enumeration, offering exact solutions with markedly reduced computation effort.

Number of windows	Number of machines	Number of testing	BNB Algorithm		EA Algorithm		% efficiency BNB-EA		
			F^a	CPU's time (sec)	F^a	CPU's time (sec)	F^a	CPU's time (sec)	Performance Index
2	2	100	43.55	0.09	43.55	0.68	100%	91.13%	66.11%
2	3	100	57.91	2.40	57.91	18.42	100%	86.99%	92.78%
2	4	100	58.93	7.87	58.93	69.71	100%	89.60%	87.46%
2	5	100	59.61	9.23	59.61	73.59	100%	90.22%	92.21%
2	6	100	63.59	66.25	63.59	247.45	100%	88.33%	96.65%
2	7	100	66.28	57.20	66.28	217.63	100%	87.82%	92.50%
2	8	100	69.98	88.01	69.98	295.64	100%	87.29%	97.51%
2	9	100	76.00	89.78	76.00	306.72	100%	88.59%	92.74%
2	10	100	81.71	89.08	81.71	320.73	100%	89.74%	93.81%
2	11	100	83.78	117.17	83.78	372.46	100%	85.59%	94.71%
						Min	100%	85.59%	66.11%
						Max	100%	91.13%	97.51%
						Mean	100%	88.53%	90.65%
						Standard Deviation	–	1.68%	9.05%

Table 12. Comparison results between the Enumeration algorithm with branch-and-bound algorithm

The next step is to compare the proposed algorithm with the Branch and Bound (BNB) method using several test cases. The objective is to obtain numerical insights into the performance of the proposed algorithm relative to the BNB method, measured in terms of minimizing the total actual flow time (F^a) and the required central processing unit (CPU) time. The results of this comparison are presented in Table 13.

From Table 13, the experiment was conducted more than 3,000 times under identical scheduling conditions to evaluate the performance of the Proposed Algorithm (PA) and the Branch-and-Bound (BNB) method. However, Table 13 only presents a representative sample of 1,000 experimental results, as all trials exhibited consistent performance patterns. As shown in the table, the average efficiency percentage of F^a obtained from the PA reached 98.25% with a standard deviation of 0.38%, while the average CPU efficiency reached 78.48% with a deviation of 15.21%. These results demonstrate that both algorithms produce consistent and high-quality solutions, with the PA showing superior computational efficiency and the BNB maintaining comparable accuracy. The minor deviations observed between runs are mainly caused by rounding operations in the mathematical model, particularly in Equation 10 for integer demand allocation and Equation 14 for integer batch size computation. Since these two equations are interdependent, small rounding differences, either upward or downward, can lead to slight variations in total actual flow time without significantly affecting the overall optimality.

Number of windows	Number of machines	Number of testing	Proposed Algorithm		BNB-Algorithm		% efficiency PA-BNB	
			F^a	CPU's time (sec)	F^a	CPU's time (sec)	F^a	CPU's time (sec)
2	2	100	44.51	0.04	43.55	0.09	98.09%	37.21%
2	3	100	58.86	0.01	57.91	2.40	98.68%	76.31%
2	4	100	59.68	0.02	58.93	7.87	98.82%	74.15%
2	5	100	61.02	0.02	59.61	9.23	97.80%	81.48%
2	6	100	64.66	0.05	63.59	66.25	98.33%	87.47%
2	7	100	67.50	0.10	66.28	57.20	98.27%	82.83%
2	8	100	71.28	0.15	69.98	88.01	98.34%	86.29%
2	9	100	77.86	0.19	76.00	89.78	97.69%	86.50%
2	10	100	83.59	0.18	81.71	89.08	97.88%	86.41%
2	11	100	85.10	0.28	83.78	117.17	98.57%	86.17%
Min							97.69%	37.21%
Max							98.82%	87.47%
Mean							98.25%	78.48%
Standard Deviation							0.38%	15.21%

Table 13. Comparison results between the proposed algorithm with branch-and-bound algorithm

As presented in Table 9 and Table 13, the comparative results highlight the consistent performance of the Proposed Algorithm (PA) when evaluated against both the Enumeration Algorithm (EA) and the Branch-and-Bound (BNB) method. Table 9 shows that the PA achieved an average efficiency of 99.32% with a deviation of 0.4% compared to the optimal solutions obtained by the EA, confirming that the proposed method is capable of generating near-optimal results with drastically reduced computation time. Meanwhile, Table 13 demonstrates that, when compared with the BNB method, the PA maintains a similar level of accuracy while executing much faster across all test conditions. Although the BNB method provides slightly more stable accuracy, it requires substantially higher computational effort. Overall, these findings indicate that the EA serves as a theoretical benchmark for optimality, the BNB offers a balanced compromise between accuracy and computational cost, and the PA provides the most practical alternative for large scale or real time scheduling problems where computational speed and scalability are crucial.

5. Conclusions

This research successfully developed a batch scheduling model for unrelated parallel machines under resource constraints and sequence-dependent setup times, aiming to minimize the total actual flow time. The proposed algorithm proved highly effective, achieving an average efficiency of 99.32% compared to the optimal solutions obtained through the Enumeration Algorithm (EA), with only a minor deviation of 0.4%. Further comparative analysis with the Branch and Bound (BNB) method demonstrated that the proposed algorithm achieved comparable solution quality while requiring significantly less computational time, whereas the BNB method provided a balanced compromise between accuracy and computational effort. Additional evaluation between the BNB and EA confirmed that both algorithms consistently reached identical optimal solutions; however, the BNB achieved this with substantially reduced computation time through an effective pruning mechanism based on Lower and Upper Bound thresholds. These findings validate that both exact algorithms ensure global optimality, while the proposed algorithm maintains near-optimal performance with exceptional computational efficiency, making it suitable for complex and large-scale scheduling environments.

Future research could extend this model to more dynamic production systems, such as flexible flowline environments where unrelated parallel machines are deployed at multiple stages. Exploring additional factors, such

as machine breakdowns or real-time job arrivals, could further enhance the model's applicability in diverse industrial scenarios.

Data Availability Statement

The data that support the findings of this study are available in https://zenodo.org/ at DOI https://doi.org/10.5281/zenodo.17373704, reference number 17373704. These data were derived from the following resources available in the public domain: https://zenodo.org/records/17373704.

The dataset does not involve human subjects, personal information, or any sensitive content, and therefore does not raise concerns related to human subject protection, ethics, privacy, or security. The authors declare no competing interests. There are no conflicts of interest with any person, institutions, or organizations related to the content of this manuscript.

Declaration of Conflicting Interests

The authors declared no potential conflicts of interest with respect to the research, authorship, and/or publication of this article.

Funding

This research was funded by the Internal Research Funding Scheme of the Research and Community Service Department, Universitas Jenderal Achmad Yani (Unjani), Indonesia, 2023, and the PPMI Scheme of Institut Teknologi Bandung (ITB), Indonesia, grant number 02/IT1.C07/SK-KP/2022.

References

Agárdi, A., & Nehéz, K. (2021). The Unrelated Parallel Machines Scheduling Problem with Machine and Job Dependent Setup Times, Availability Constraints, Time Windows and Maintenance Times. *Management and Production Engineering Review*, 12(3), 15-24. Available at: https://api.semanticscholar.org/CorpusID:238863350

Alhajjar, A., & Mohammed, H. (2023). Some Algorithms Used in Parallel Machine Scheduling. *Wasit Journal for Pure Sciences*, 2. https://doi.org/10.31185/wjps.158

Baker, K.R., & Trietsch, D. (2009). Safe scheduling: Setting due dates in single-machine problems. *European Journal of Operational Research*, 196(1), 69-77. https://doi.org/10.1016/j.ejor.2008.02.009

Cheng, T.C.E., & Sin, C.C.S. (1990). A state-of-the-art review of parallel-machine scheduling research. *European Journal of Operational Research*, 47(3), 271-292. https://doi.org/10.1016/0377-2217(90)90215-W

Foroutan, R.A., Shafipour, M., Rezaeian, J., & Khojasteh, Y. (2024). Just-in-time scheduling of unrelated parallel machines with family setups and soft time window constraints. *Journal of Industrial and Production Engineering*, 41(8), 692-715. https://doi.org/10.1080/21681015.2024.2361046

Georgiadis, G.P., Elekidis, A.P., & Georgiadis, M.C. (2019). Optimization-based scheduling for the process industries: From theory to real-life industrial applications. *Processes*, 7(7). https://doi.org/10.3390/pr7070438

Goli, A., & Keshavarz, T. (2022). Just-in-time scheduling in identical parallel machine sequence-dependent group scheduling problem. *Journal of Industrial & Management Optimization*, 18(6), 3807-3830. Available at: https://api.semanticscholar.org/CorpusID:237959397

Halim, A.H., Miyazaki, S., & Ohta, H. (1991). A Batch-Scheduling Problem to Minimize Actual Flow Times on Hetero geneous Machines under JIT Environment. *Bulletin of the University of Osaka Prefecture*, 40(A), 7137.

Halim, A.H., Miyazaki, S., & Ohta, H. (1994a). Batch-scheduling problems to minimize actual flow times of parts through the shop under JIT environment. *European Journal of Operational Research*, 72(3), 529-544. https://doi.org/10.1016/0377-2217(94)90421-9

Halim, A. H., Miyazaki, S., & Ohta, H. (1994b). Lot Scheduling Problems of Multiple Items in The Shop with Both Receiving and Delivery Just In Time. *Production and Planning Control,* 5(2), 175-184. https://doi.org/10.1080/09537289408919484

Kazemi, H., Mahdavi-Mazdeh, M., Rostami, M., & Heydari, M. (2021). The integrated production-distribution scheduling in parallel machine environment by using improved genetic algorithms. *Journal of Industrial and Production Engineering*, 38(3), 157-170. https://doi.org/10.1080/21681015.2020.1848930

Khanh-Van, B., & Van-Hop, N. (2021). Genetic algorithm with initial sequence for parallel machines scheduling with sequence dependent setup times based on earliness- tardiness. *Journal of Industrial and Production Engineering*, 38(1), 18-28. https://doi.org/10.1080/21681015.2020.1829111

Kong, M., Liu, X., Pei, J., Pardalos, P.M., & Mladenovic, N. (2020). Parallel-batching scheduling with nonlinear processing times on a single and unrelated parallel machines. *Journal of Global Optimization*, 78(4), 693-715. https://doi.org/10.1007/s10898-018-0705-3

Kurniawan, D., Yusriski, R., Isnaini, M.M., Anas, M., & Halim, A.H. (2021). A Flow Shop Batch Scheduling Model with Part Deterioration and Operator Learning-Forgetting Effects to Minimize Total Actual Flow Time. *Proceedings of the Second Asia Pacific International Conference on Industrial Engineering and Operations Management*, 1936, 644-655.

Kurniawan, D., Yusriski, R., Isnaini, M.M., Ma'Ruf, A., & Halim, A.H. (2024). A Flow Shop Batch Scheduling Model with Pre-Processing and Time- Changing Effects to Minimize Total Actual Flow Time. *Journal of Industrial Engineering and Management*, 17(2), 542-561.

Lee, J. H., & Jang, H. (2019). Uniform Parallel Machine Scheduling with Dedicated Machines, Job Splitting and Setup Resources. *Sustainability (Switzerland)*, *11*(24), 1–23. https://doi.org/10.3390/su11247137

Li, K., Zhang, X., Leung, J.Y.T., & Yang, S.L. (2016). Parallel machine scheduling problems in green manufacturing industry. *Journal of Manufacturing Systems*, 38, 98-106. https://doi.org/10.1016/j.jmsy.2015.11.006

Maulidya, R., Suprayogi, Wangsaputra, R., & Halim, A.H. (2020). A batch scheduling model for a three-stage hybrid flowshop producing products with hierarchical assembly structures. *International Journal of Technology*, 11(3), 608-618. https://doi.org/10.14716/ijtech.v11i3.3555

Miao, C., Zhang, Y., & Cao, Z. (2011). Bounded parallel-batch scheduling on single and multi machines for deteriorating jobs. *Information Processing Letters*, 111(16), 798-803. https://doi.org/10.1016/j.ipl.2011.05.018

Mokotoff, E., Jimeno, J.L., & Gutiérrez, A.I. (2001). List scheduling algorithms to minimize the makespan on identical parallel machines. *Top*, 9(2), 243-269. https://doi.org/10.1007/bf02579085

Muñoz-Díaz, M.L., Escudero-Santana, A., & Lorenzo-Espejo, A. (2024). Solving an Unrelated Parallel Machines Scheduling Problem with machine- and job-dependent setups and precedence constraints considering Support Machines. *Computers and Operations Research*, 163. https://doi.org/10.1016/j.cor.2023.106511

Muñoz-Villamizar, A., Santos, J., Montoya-Torres, J., & Alvaréz, M.J. (2019). Improving effectiveness of parallel machine scheduling with earliness and tardiness costs: A case study. *International Journal of Industrial Engineering Computations*, 10(3), 375-392. https://doi.org/10.5267/j.ijiec.2019.2.001

Olteanu, A.L., Sevaux, M., & Ziaee, M. (2022). Unrelated Parallel Machine Scheduling with Job and Machine Acceptance and Renewable Resource Allocation. *Algorithms*, 15(11), 1-17. https://doi.org/10.3390/a15110433

Senthilkumar, B., Kannan, T., & Madesh, R. (2017). Optimization of flux-cored arc welding process parameters by using genetic algorithm. *International Journal of Advanced Manufacturing Technology*, 93(1-4), 35-41. https://doi.org/10.1007/s00170-015-7636-7

Shahvari, O., & Logendran, R. (2017). An Enhanced tabu search algorithm to minimize a bi-criteria objective in batching and scheduling problems on unrelated-parallel machines with desired lower bounds on batch sizes. *Computers and Operations Research*, 77, 154-176. https://doi.org/10.1016/j.cor.2016.07.021

Shahvari, O., Logendran, R., & Tavana, M. (2022). An efficient model-based branch-and-price algorithm for unrelated-parallel machine batching and scheduling problems. *Journal of Scheduling*, 25(5), 589-621. https://doi.org/10.1007/s10951-022-00729-7

Xu, J., & Nagi, R. (2013). Identical parallel machine scheduling to minimise makespan and total weighted completion time: A column generation approach. *International Journal of Production Research*, 51(23-24), 7091-7104. https://doi.org/10.1080/00207543.2013.825379

Yusriski, R., Astuti, B., Biksono, D., & Wardani, T.A. (2021). A single machine multi-job integer batch scheduling problem with multi due date to minimize total actual flow time. *Decision Science Letters*, 10(3), 231-240. https://doi.org/10.5267/j.dsl.2021.4.002

Yusriski, R., Astuti, B., Ilham, M., & Zahedi (2019). Integrated Batch Production and Multiple Preventive Maintenance Scheduling on A Single Machine to Minimize Total Actual Flow Time. *IOP Conference Series: Materials Science and Engineering*, 598(1), 12083. https://doi.org/10.1088/1757-899X/598/1/012083

Yusriski, R., Sukoyo, Samadhi, T.M.A.A., & Halim, A.H. (2015). Integer batch scheduling problems for a single-machine with simultaneous effects of learning and forgetting to minimize total actual flow time. *International Journal of Industrial Engineering Computations*, 6(3), 365-378. https://doi.org/10.5267/j.ijiec.2015.2.005

Yusriski, R., Sukoyo, Samadhi, T.M.A.A., & Halim, A.H. (2016). An Integer Batch Scheduling Model for a Single Machine with Simultaneous Learning and Deterioration Effects to Minimize Total Actual Flow Time. *IOP Conference Series: Materials Science and Engineering*, 114(1). https://doi.org/10.1088/1757-899X/114/1/012073

Yusriski, R., Sukoyo, Samadhi, T.M.A.A., & Halim, A.H. (2018). An integer batch scheduling model considering learning, forgetting, and deterioration effects for a single machine to minimize total inventory holding cost. *IOP Conference Series: Materials Science and Engineering*, 319(1). https://doi.org/10.1088/1757-899X/319/1/012038

Zarandi, M.H.F., & Kayvanfar, V. (2015). A bi-objective identical parallel machine scheduling problem with controllable processing times: a just-in-time approach. *International Journal of Advanced Manufacturing Technology*, 77(1-4), 545-563. https://doi.org/10.1007/s00170-014-6461-8

Appendix

Constraint (3) shows that shows that the demand of jobs can be divided into several sub-demands in each machine. Let be assumed that the number of batches in each machine ($N_{[k]m}$) equal to 1, the batch sizes is a number of units that allocated to the machine, notated by A_{km}. Constraint (3) can be written as follows

$$\Delta_{[k]m} + t_{[k]m}A_{[k]m} + s_{[k]m} \leq d_{[k]}; k = 1, \dots, K;\ m = 1, .., M; \tag{A.1}$$

Since $d_{[k]}$ is similar for each m so Equation (A1) can be written as follows.

$$\begin{aligned}\Delta_{[k]1} + t_{[k]1}A_{[k]1} + s_{[k]1} = \Delta_{[k]2} + t_{[k]2}A_{[k]2} + s_{[k]2} = \cdots \\ = \Delta_{[k]M-1} + t_{[k]M}A_{[k]M-1} + s_{[k]M-1} = \Delta_{[k]M} + t_{[k]M}A_{[k]M} + s_{[k]M}\end{aligned} \tag{A.2}$$

For $M = 2$, Equation (A.2) can be expand as follows.

$$\Delta_{[k]1} + t_{[k]1}A_{[k]1} + s_{[k]1} = \Delta_{[k]2} + t_{[k]2}A_{[k]2} + s_{[k]2} \tag{A.3}$$

Solving $A_{[k]1}$ from Equation (A.3) found as follows.

$$A_{[k]1} = \frac{\Delta_{[k]2} - \Delta_{[k]1} + t_{[k]2}A_{[k]2} + s_{[k]2} - s_{[k]1}}{t_{[k]1}} \tag{A.4}$$

The formula of $A_{[k]2}$ as follows.

$$A_{[k]2} = \frac{\Delta_{[k]1} - \Delta_{[k]2} + t_{[k]1}A_{[k]1} + s_{[k]1} - s_{[k]2}}{t_{[k]2}} \tag{A.5}$$

From Constrain (3) found as follows.

$$A_{[k]1} = n_{[k]} - A_{[k]2} \tag{A.6}$$

Substituting Equation (A.5) into (A.6) found as follows.

$$A_{[k]1} = n_{[k]} - \frac{\Delta_{[k]1} + s_{[k]1} + t_{[k]1}A_{[k]1} - \Delta_{[k]2} - s_{[k]2}}{t_{[k]2}} \tag{A.7}$$

Solving $A_{[k]1}$ from Equation (A.7) found as follows.

$$A_{[k]1} = \frac{t_{[k]2}n_{[k]} + \Delta_{[k]2} + s_{[k]2} - \Delta_{[k]1} - s_{[k]1}}{t_{[k]1} + t_{[k]2}} \tag{A.8}$$

Using similar step to solve $A_{[k]1}$, the $A_{[k]2}$ can be solved as follows.

$$A_{[k]2} = \frac{t_{[k]1}n_{[k]} + \Delta_{[k]1} + s_{[k]1} - \Delta_{[k]2} - s_{[k]2}}{t_{[k]1} + t_{[k]2}} \tag{A.9}$$

For $M = 3$, the formula of $A_{[k]1}$, $A_{[k]2}$ and $A_{[k]3}$, as follows.

$$A_{[k]1} = \frac{\Delta_{[k]2} - \Delta_{[k]1} + t_{[k]2}A_{[k]2} + s_{[k]2} - s_{[k]1}}{t_{[k]1}} \tag{A.10}$$

$$A_{[k]1} = \frac{\Delta_{[k]3} - \Delta_{[k]1} + t_{[k]3}A_{[k]3} + s_{[k]3} - s_{[k]1}}{t_{[k]3}} \tag{A.11}$$

$$A_{[k]2} = \frac{\Delta_{[k]1} - \Delta_{[k]2} + t_{[k]1}A_{[k]1} + s_{[k]1} - s_{[k]2}}{t_{[k]2}} \tag{A.12}$$

$$A_{[k]2} = \frac{\Delta_{[k]3} - \Delta_{[k]2} + t_{[k]3}A_{[k]3} + s_{[k]3} - s_{[k]2}}{t_{[k]2}} \tag{A.13}$$

$$A_{[k]3} = \frac{\Delta_{[k]1} - \Delta_{[k]3} + t_{[k]1}A_{[k]1} + s_{[k]1} - s_{[k]3}}{t_{[k]3}} \tag{A.14}$$

$$A_{[k]3} = \frac{\Delta_{[k]2} - \Delta_{[k]3} + t_{[k]2}A_{[k]2} + s_{[k]2} - s_{[k]3}}{t_{[k]3}} \tag{A.15}$$

From Equation (2) we found:

$$A_{[k]1} = n_{[k]} - A_{[k]2} - A_{[k]3} \tag{A.16}$$

Substituting Equation (A.12) and (A.14) into (A.16) found as follows.

$$A_{[k]1} = n_{[k]} - \frac{\Delta_{[k]1} - \Delta_{[k]2} + t_{[k]1}A_{[k]1} + s_{[k]1} - s_{[k]2}}{t_{[k]2}} - \frac{\Delta_{[k]1} - \Delta_{[k]3} + t_{[k]1}A_{[k]1} + s_{[k]1} - s_{[k]3}}{t_{[k]3}} \tag{A.17}$$

solving $A_{[k]1}$ found as follows.

$$A_{[k]1} = \frac{n_{[k]}t_{[k]2}t_{[k]3} - t_{[k]3}(\Delta_{[k]1} - \Delta_{[k]2} + s_{[k]1} - s_{[k]2}) - t_{[k]2}(\Delta_{[k]1} - \Delta_{[k]3} + s_{[k]1} - s_{[k]3})}{t_{[k]1}t_{[k]2} + t_{[k]1}t_{[k]3} + t_{[k]2}t_{[k]3}} \tag{A.18}$$

Re-writing Equation (A.18) can be found as follows.

$$A_{[k]1} = \frac{\prod_{j=1}^{M} t_{[k]j}\{n_{[k]} + \sum_{i=1}^{M}(\Delta_{[k]i} - \Delta_{[k]1} + s_{[k]i} - s_{[k]1})t_{[k]i}^{-1}\}}{t_{[k]1}\prod_{j=1}^{M} t_{[k]j}\sum_{i=1}^{M} t_{[k]i}^{-1}} \tag{A.19}$$

Simplifying Equation (A.19) found as follows.

$$A_{[k]1} = \frac{n_{[k]} + V}{t_{[k]1}W} - \frac{(\Delta_{[k]1} + s_{[k]1})}{t_{[k]1}};$$
$$\text{where } V = \sum_{i=1}^{M}\left(\frac{\Delta_{[k]i} + s_{[k]i}}{t_{[k]i}}\right) \text{ and } W = \sum_{i=1}^{M}\left(\frac{1}{t_{[k]i}}\right),\ i \in m \tag{A.20}$$

Base on Equation (A.20),

the general formula for $m = 1, \ldots, M$ as follows.

$$A_{[k]m} = \frac{n_{[k]} + V}{t_{[k]m}W} - \frac{(\Delta_{[k]m} + s_{[k]m})}{t_{[k]m}};$$
$$\text{where } V = \sum_{i=1}^{M}\left(\frac{\Delta_{[k]i} + s_{[k]i}}{t_{[k]i}}\right) \text{ and } W = \sum_{i=1}^{M}\left(\frac{1}{t_{[k]i}}\right),\ i \in m \ \blacksquare \tag{A.21}$$