Yugoslav Journal of Operations Research
(20##), Number #, #-#
DOI: https://doi.org/10.2298/YJOR240415039A

Research article

OPTIMIZING MANUFACTURING EFFICIENCY: AN EVALUATION OF HEURISTIC ALGORITHMS FOR NON-PREEMPTIVE FLOW-SHOP SCHEDULING WITH MAKESPAN CRITERION

Md. Kawsar Ahmed ASIF*

Department of Mathematics, University of Dhaka, Bangladesh,
Department of General Education, Canadian University of Bangladesh, Bangladesh,
ahmedasif.math.du@gmail.com, ORCID: 0000-0002-8295-0522

Sohana JAHAN

Department of Mathematics, University of Dhaka, Bangladesh,
sjahan.mat@du.ac.bd, ORCID: 0000-0002-6885-9511

Md. Rajib AREFIN

Department of Mathematics, University of Dhaka, Bangladesh,
arefin.math@du.ac.bd, ORCID: 0000-0002-4696-6877

Faria TABASSUM

Department of Computer Science and Engineering, Bangladesh University of Engineering and Technology, Bangladesh,
fariatabassum3@gmail.com, ORCID: 0009-0002-2354-7052

Received: April 2025 / Accepted: September 2025

Abstract: In today's competitive and technology-driven industrial landscape, optimizing manufacturing efficiency remains a primary objective. Production scheduling emerges as a critical decision-making tool in achieving this goal, playing a pivotal role in coordinating manufacturing operations. Over the past few decades, researchers have developed numerous exact, heuristic, and meta-heuristic scheduling algorithms for diverse production environments. However, selecting a compatible algorithm remains a significant challenge due to the combinatorial nature of scheduling, further compounded by the NP-completeness of the non-preemptive flow-shop scheduling problem. Therefore, this study evaluates eight heuristic algorithms for solving the non-preemptive flow-shop scheduling problem, with a focus on makespan minimization. Using empirical data from a real-world

*Corresponding author

Ready-Made Garment (RMG) manufacturing facility, the performance of the heuristics is assessed in terms of makespan, idle time, and time complexity. Among the methods analyzed, the NEH heuristic demonstrates the most favorable result, with a 2.91% deviation from the theoretical lower bound. The study provides practical insights for researchers and practitioners in selecting effective heuristics under varying scheduling conditions.

Keywords: Flow-Shop, heuristics, makespan, NP-complete, optimal sequence, RMG, scheduling, time complexity.

MSC: 90B35, 68M20, 90C59

1. INTRODUCTION

Scheduling is the strategic component in determining the cost of products or services in the manufacturing and services industries, and it helps the industries maximize productivity using minimum resources. In operations research and computer science, flow-shop scheduling is an optimization problem and a variant of optimum job scheduling where a set of jobs must be processed on specified machines according to a given order to minimize the total elapsed time and idle time. When n jobs or activities are assigned to be processed on m machines or resources in a flow-shop scheduling environment, there will be $(n!)^m$ number of possible sequences. However, only the optimal sequence minimizes the total completion time (makespan) and idle time [1]. Therefore, the scheduling problem deal with determining an appropriate sequence of jobs for a given number of machines.

The time and space complexity of the Flow Shop Sequencing and Scheduling Problem (FSSP) plays a significant role in algorithm selection. With $(n!)^m$ potential sequences, exhaustive search quickly becomes impractical. While exact solutions might not be achievable in polynomial time, heuristic algorithms strive to strike a balance between solution quality and computational efficiency. It can be challenging to choose the best sequence for the flow-shop scheduling problem (FSP). In fact, the problem actually falls within the category of NP-complete problems; hence it is unlikely that there is a polynomial-time method that can solve it accurately every time [2]. NP-complete problems are "hard" because they take a long time to solve, even for input sizes that are comparatively small. A problem is considered to be NP-complete if it belongs to the set of NP (nondeterministic polynomial time) problems and can be reduced to by every other NP problem in polynomial time [3]. FSSP is known to be NP-complete. This is why, various heuristic and metaheuristic algorithms have been devised to find approximate solutions. In order to optimize some performance metric such as time, cost, or weight, the scheduling problem aims to determine the order or sequence in which the machines will execute the jobs. The scheduling problem's efficacy can be assessed in terms of meeting due dates, maximizing profits, minimizing time, and minimizing cost. Scheduling models require resources (machines) and tasks (jobs) to function. The system is referred to as static when the list of tasks that are available for scheduling remains constant over time, as opposed to dynamic situations where new jobs keep arriving over time [4].

Most often, the terms 'Scheduling' and 'Sequencing' are used synonymously. However, the distinction between these two terms is essential as both are related to the flow-shop and job-shop scheduling problem. According to Conway et al., there will always be a sequencing problem whenever there is an option regarding the order in which a number of jobs can be executed [5]. As defined by Baker, the sequencing problem is a specific case

of scheduling where the job order (in which tasks are completed) totally regulates the schedule [6]. In general, the order in which the machines process the jobs is known as sequencing. The allocation of machines across time to complete a set of jobs, on the other hand, is known as scheduling [4].

Manufacturing and services industries seamlessly deal with the different scheduling environments. By understanding and distinguishing the various scheduling environments, operations researchers and operation managers can develop and implement scheduling algorithms that enable companies to tailor their strategies, processes, and resource allocation to specific needs and, consequently, can help industries to improve their efficiency, productivity, and profitability. Here are some specific examples of how comprehending and differentiating different scheduling environments can be applied in the manufacturing and services industries:

- A car manufacturer might use a flow shop scheduling algorithm to schedule the production of cars on a production line.
- A hospital might use a job shop scheduling algorithm to schedule the treatment of patients.
- A restaurant might use a batch scheduling algorithm to schedule the preparation of meals.
- A shipping company might use a parallel machine scheduling algorithm to schedule the loading and unloading of ships.

The scheduling environment is the set of constraints that the jobs and machines must satisfy. The appropriate algorithms are those that have been shown to be effective in scheduling jobs in that environment. The following Table 1 shows the different scheduling environments and the appropriate algorithms that can be used to schedule jobs in those environments.

Table 1: An overview of various scheduling environments and the notable scheduling algorithms

Scheduling Environment	***Key Characteristics***	***Examples of Applications***	***Notable Scheduling Algorithm (Author(s), Year)***
Flow Shop Scheduling	Series of machines or workstations arranged in a fixed sequence.	Production lines, assembly lines, manufacturing processes, printing	Johnson's Rule (S. M. Johnson, 1954)
			Palmer's Heuristic (D. S. Palmer, 1965)
			CDS (Campbell, Dudek & Smith, 1970)
			NEH (Nawaz, Enscore & Ham, 1983)
Job Shop Scheduling	Various machines with different processing requirements.	Custom manufacturing, repair shops, tool and die shops, aerospace	Branch and Bound (Ignall & Schrage, 1965)
			Shifting Bottleneck Heuristic (Baker, 1974)
			Genetic Algorithm (J. Holland, 1975)
			Tabu Search (F. Glover, 1986)
Permutation Flow Shop Scheduling	Multiple flow shops with different orders and processing times.	Semiconductor manufacturing, PCB assembly, automobile production	Gupta's Algorithm (J. N. D. Gupta, 1971)
			NEH (Nawaz, Enscore & Ham, 1983)
			Cyclic Descent Search (Hsu & Lee, 1989)
Open Shop Scheduling	No specific sequence for jobs or machines; flexibility in job-machine assignments.	Job scheduling in service industries, job allocation, task scheduling	Earliest Due Date Rule (Weiss, 1962)
			Priority Rule (Schrage, 1968)
			Genetic Algorithm (J. Holland, 1975)
Parallel Machine Scheduling	Multiple identical machines capable of processing jobs simultaneously.	Parallel computing, data center task scheduling, cloud computing	List Scheduling (Graham, 1966)
			Genetic Algorithm (J. Holland, 1975)
			Max-Min Algorithm (Lenstra, 1977)

Table 1: (continued)

Scheduling Environment	*Key Characteristics*	*Examples of Applications*	*Notable Scheduling Algorithm (Author(s), Year)*
Unrelated Machine Scheduling	Different machines with varying capabilities for processing jobs.	Job scheduling in machine shops, semiconductor manufacturing, metalworking	Critical Ratio Scheduling (Berry & Rao, 1968)
			Simulated Annealing (Kirkpatrick, 1983)
			Makespan Minimization (Liao & Lin, 2008)
Batch Scheduling	Grouping of similar jobs to optimize processing efficiency.	Chemical processing, food production, pharmaceutical manufacturing	Economic Lot Scheduling (Harris, 1913)
			Genetic Algorithm (J. Holland, 1975)
			Shortest Processing Time (L. Pinedo, 1995)
Dynamic Scheduling	Constantly changing job arrival times and other dynamic factors.	Real-time production, traffic management, healthcare scheduling	Earliest Deadline First (Liu & Layland, 1973)
			Least Slack Time First (Baker, 1974)
			Weighted Fair Queuing (A. Parekh, 1993)

By understanding how these algorithms handle larger instances of flow-shop scheduling problems, we can gain insights into their practicality for real-world scheduling challenges. The necessary notations and parameters are listed in Table 2 and will be used throughout this article.

Table 2: List of notations and parameters used

Symbols and Notations	*Description*
J	Set of jobs, J $= \{1, 2, 3, \dots, n\}$, assigned by j
M	Set of machines, M $= \{1, 2, 3, \dots, m\}$, assigned by i
n	Number of jobs to be processed
m	Number of machines in a shop
$O(j, i)$ or t_{ij} or p_{ij}	Processing time of job j on machine i
σ	Optimal sequence of jobs
$S(j)$	Slope index for each job or the weight assigned to job j
$W(j)$	Weight of job j
T_j	Cumulative processing time for each individual job
C_{max} or MS	Maximum completion time or Makespan
LB	Lower Bound (LB) of the Makespan C_{max}
A_j	Binary indicator (either 1 or -1) for each job j
$f(i)$	The function associated with job i
T_{ij}^k	Cumulative processing time up to the k-th order for the i job and j machine
$\mathrm{O}(f(n))$	Big O notation, indicating the maximum growth rate of a function $f(n)$ in terms of the input size n.

The novelty of this study lies in its systematic evaluation of eight classical heuristic algorithms within a real-world flow-shop scheduling context derived from a ready-made garment manufacturing unit. Unlike prior research, which typically focuses on one or two heuristics or omits performance metrics beyond makespan, this paper provides a multi-faceted comparison that includes idle time and computational complexity. Moreover, less-studied heuristics such as Modrak's, Time Deviation, RC are benchmarked alongside widely used strategies like NEH, CDS, and Palmer's, filling a distinct gap in applied scheduling research. This empirical contribution is particularly valuable for mid-scale deterministic environments that require low-overhead, high-efficiency scheduling solutions.

This paper will examine various heuristic approaches to solve the scheduling problems with the makespan minimization criterion in a flow-shop environment. The subsequent part of this article is organized as follows: Section 2 provides a comprehensive review of prior research contributions by various authors in the field of flow shop scheduling, focusing on makespan-related approaches and findings. Section 3 formally defines the flow shop scheduling problem and the makespan criterion. Section 4 details the methodology, discussing eight prominent heuristic algorithms and their time complexity. Section 5 presents an empirical comparison among the chosen heuristic algorithms. Real-world data from a local Ready-Made Garments (RMG) manufacturing company is employed for this comparative analysis. Section 6 highlights the core findings of the research, revealing the outcomes of the comparative analysis. Additionally, this section conducts a comprehensive complexity analysis of each algorithm, offering insights into computational efficiency and scalability. Finally, Section 7 summarizes the main conclusions drawn from the research and offers future recommendations in this critical domain of operations research.

This study distinguishes itself by offering a comparative evaluation of eight classical heuristics in a real-world RMG context, analyzing not only makespan but also idle time and computational complexity. The analysis includes the rarely tested Modrak and RA heuristics in flow-shop environments, which has not been emphasized in prior literature.

2. LITERATURE REVIEW

A review of the literature highlights that over the last seven decades, most heuristics aimed at minimizing makespan in flow shop scheduling. The Johnson (1954) heuristic was among the earliest [7], focusing on the two-machine flow shop; later, researchers developed diverse heuristics for more than two machine scenarios.

Palmer (1965) was one of the very first to introduce a heuristic which employed a slope order index to prioritize job sequencing on machines [8], favoring jobs with increasing processing times across machines, commonly known as Palmer's heuristic. Campbell, Dudek, and Smith (1970) presented an extended heuristic derived from Johnson's algorithm [9], tailored for flow shop scheduling problems focused on minimizing makespan.

Gupta (1971) proposed a functional heuristic algorithm similar to Palmer's approach [10], but with a distinct slope index formulation that incorporated insights from Johnson's rule optimality in three-machine scenarios. Dannenbring (1977) introduced a rapid access (RA) method that combines Palmer's and the CDS technique with an aim to easily generate a quality solution [11]. Rather than solving $(m-1)$ artificial two-machine problems, it tackles only one artificial problem where processing times are derived from a predefined waiting scheme.

The Nawaz-Enscore-Ham (NEH) (1983) heuristic operates on the basis that jobs with greater total processing times across all machines should be prioritized over those with lower totals [12]. NEH constructs the ultimate sequence progressively by iteratively incorporating jobs and seeking optimal partial solutions.

Rajendran and Chaudhuri (RC) (1995) introduced a heuristic with multiple optimization goals: makespan, total flow time, and machine idle time [13]. Improving upon existing heuristics, the initial sequence came from the CDS algorithm. Koulamas (1998) introduced a simple constructive heuristic called HFC for the flow-shop scheduling problem with makespan objective, capable of producing non-permutation schedules when appropriate, based on the order of jobs in two-machine problems embedded in the flow-

shop problem [14]. HFC outperforms NEH on problems where a non-permutation schedule may be optimal, indicating its potential for improving scheduling efficiency in such scenarios.

Modrak and Pandian (2010) proposed an algorithm called the MOD heuristic based on converting a multi-machine problem to a two-machine scheduling problem [15]. The algorithm is a competitive alternative for practical application as it converts the m-machine problem to a 2-machine problem, which can be solved using Johnson's algorithm. This proposed algorithm effectively finds optimal or near-optimal sequences for flow shop scheduling problems. Rao et al. (2013) introduced a novel technique called the time deviation method [16], utilizing it to derive job sequences and calculate minimum total elapsed time. Lin et al. (2017) devised a method to combine ten heuristic rules to form low-level heuristics (LLHs) within a backtracking search framework [17]. Their proposed algorithm outperforms current ones according to computational assessments on standard benchmarks.

Yang and Li (2022) addresses the distributed assembly blocking FSSP by proposing a knowledge-driven constructive heuristic algorithm with an aim to minimize maximum assembly completion time [18]. The employ three different neighborhood knowledge to enhance job assignment and assembly sequence determination across multiple machines. Asif et al. (2022) empirically analyzed exact algorithms for solving the non-preemptive flow-shop scheduling problem [4], evaluating their makespan values and goodness factor by the lower bound. The algorithms considered were Johnson's, Branch & Bound, Kusiak's, and SAI. Among these, Branch & Bound generated the highest number of sequences with near-optimal solutions, while the other algorithms were simpler but comparatively less accurate.

While classical heuristics have been widely explored, recent studies have focused on soft computing and learning-based strategies. Pan et al. (2022) introduced a knowledge-based two-population optimization (KTPO) method for the optimization of distributed energy-efficient scheduling of parallel machines [19]. The method combines factory and machine assignment decisions with domain-specific heuristics, cooperative evolution processes, and a knowledge-based local search to improve both energy consumption and tardiness.

On this basis, Zhao et al. (2023) developed a hyperheuristic embedded with Q-learning to solve multi-objective energy-efficient distributed blocking flow shop scheduling problems [20]. The approach utilizes performance feedback to choose low-level heuristics dynamically and adopts job acceleration and deceleration strategies to reduce energy consumption and tardiness. Afterwards, Zhao et al. (2024) put forward an optimized iterative greedy algorithm based on Q-learning for distributed no-idle permutation flow shop scheduling [21]. The algorithm adapts neighbourhood structures to the problem size and uses a destruction-reconstruction process with learned weighting strategies to enhance objective balancing. Jerbi et al. (2025) have recently proposed a multi-objective mathematical programming approach tailored to healthcare logistics, aiming to minimize the makespan, patient flow time, and physician workload variability [22]. Their model incorporated dispatching rules with exact optimization for small-scale problems and genetic algorithms for large-scale problems, demonstrating its practicality in real-world applications in low-resource environments.

A classification of these scheduling algorithms, focusing on the makespan objective, is illustrated in Figure 1.

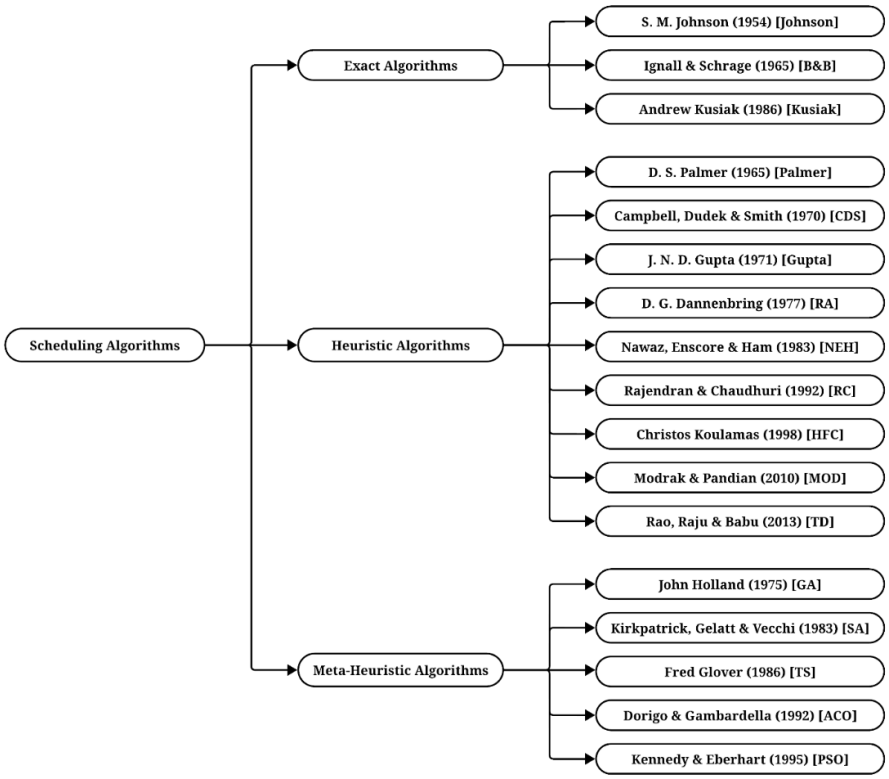

Figure 1: Classification of scheduling algorithms with makespan objective

These studies emphasize dynamic and energy-aware contexts. Our study, in contrast, focuses on deterministic scheduling within the small-to-medium scale, static environments common to many manufacturing facilities.

3. FORMULATION OF FLOW-SHOP SCHEDULING PROBLEM

The flow shop scheduling problem (FSP) is a special case of the job shop scheduling problem where all jobs have the same processing order across all machines. The FSP can be formulated as follows:

Input Parameters: A set of n jobs $J = \{J_1, J_2, J_3, \dots, J_n\}$, a set of m machines $M = \{M_1, M_2, M_3, \dots, M_m\}$, and a processing time matrix $P = [p_{ij}]$, where p_{ij} is the processing time of job J_j on machine M_i.

Objective Function: Find a sequence of jobs $\sigma = (\sigma_1, \sigma_2, \sigma_3, \dots, \sigma_n)$ that minimizes a given objective function, such as the makespan C_{max} or the total flow-time $\sum f_i$. The makespan, C_{max}, is the completion time of the last job to finish processing on all machines. The total flow-time, $\sum f_i$, is the sum of the completion times of all jobs on all machines.

Constraints:

- **Machine Constraint**: Each machine can process only one job at a time.
- **Job Constraint**: Each job must be processed exactly once.
- **Precedence Constraint**: Jobs must be processed in a fixed order.

The FSP is a complex problem that is NP-hard, meaning that it is unlikely that there exists a polynomial-time algorithm for finding the optimal solution. Therefore, heuristic algorithms are often used to solve FSP instances. Consider an instance of a 2-job & 2-machine flow shop scheduling problem is illustrated in Table 3.

Table 3: A flow shop scheduling problem with 2 jobs and 2 machines

p_{ij}	Machine (i)	
Job (j)	M_1	M_2
J_1	5	7
J_2	6	8

Each job must be assigned to M_1 first, then M_2, without changing the order. In other words, the machines are used in the same order ($M_1 \rightarrow M_2$) to execute these jobs.

4. METHODOLOGY

In order to enhance the consistency and efficiency of the machines and to achieve the optimum level of production, the sequencing and scheduling problems are analyzed. A real-world scenario has been adopted in order to examine the makespan minimization for a 5-job and 3-machine flow-shop scheduling problem. In the comparative analysis, data collected from a nearby manufacturer of ready-made garments (RMG) is presented. On a variety of jobs (items), the manufacturer must carry out the three primary operations of cutting, sewing, and packaging. Each job has a known completion time for these processes, measured in minutes. With the objective to identify the best optimal outcome in a flow-shop environment for the effective operation of the machine, we will now apply Palmer's heuristic, CDS heuristic, Gupta's heuristic, RA heuristic, NEH heuristic, RC heuristic, Modrak's heuristic, and Time Deviation method algorithms.

These heuristics were selected based on their strategic diversity and computational relevance: Palmer's uses slope-based priority indexing; CDS iteratively extends Johnson's rule; Gupta's offers rule-based decision simplification; RA incorporates slope-weighted ranking logic; NEH builds optimal sequences through progressive job insertion; RC integrates weighted job positioning; Modrak emphasizes load balancing through ranking; and Time Deviation aims to minimize time dispersion. Collectively, these heuristics represent a wide spectrum of classical, hybrid, and balance-oriented scheduling strategies, allowing for a comprehensive analysis across different job-machine dynamics without the complexities of stochastic or metaheuristic frameworks.

To validate the effectiveness of the selected heuristic algorithms, we compared their makespan results against a theoretical lower bound derived from the minimum cumulative processing time. Additionally, to ensure generalizability and robustness, each heuristic was tested under multiple problem configurations, including variations in job order and machine sequence. Benchmarking was further performed against selected instances from standard FSP datasets (e.g., Taillard instances), which allowed us to assess consistency

across different scenarios. The comparative findings confirmed the internal validity of the model and helped isolate the practical strengths of each heuristic under varying shop-floor conditions.

4.1. Palmer's Heuristic

Palmer's heuristic algorithm, also known as the Slope Index (SI) method, is an approach used to solve job scheduling problems involving n jobs and m machines. The algorithm aims to find an efficient sequence for scheduling jobs on machines by assigning specific weights to each machine and evaluating the weights of individual jobs based on a given procedure. The goal is to minimize the makespan, which represents the completion time of the entire schedule. Here are the steps involved in Palmer's heuristic algorithm:

Step 1. Define a job scheduling problem with n jobs and m machines.

Step 2. Assign weights to each machine using the formula:

$$S(j) = -\sum_{i=1}^{m}[k-(2i-1)] \cdot O(j,i) \quad (1)$$

Where,

- $S(j)$ represents the weight assigned to job j.
- k is a constant term derived from the number of machines m.
- i iterates over the machines from 1 to m.
- $O(j,i)$ denotes the processing time of job j on machine i.

Step 3. Evaluate the weight of each job using the procedure described in *Step 2* by multiplying the weights assigned to the machines with their corresponding processing times:

$$W(j) = S(j) \cdot \sum_{i=1}^{m} O(j,i) \quad (2)$$

Here,

- $W(j)$ stands for the weight of job j.
- $S(j)$ corresponds to the machine weight for job j as determined in *Step 2*.
- $O(j,i)$ signifies the processing time of job j on machine i.

Step 4. Arrange the jobs in descending order based on their computed weights $W(j)$.

Step 5. Formulate a job sequence based on the sorting performed in *Step 4*. This sequence represents the order in which jobs will be scheduled on the machines.

Step 6. Compute the makespan for the obtained job sequence. The makespan represents the total time required for the entire scheduling process, encompassing the completion times of all jobs.

In summary, Palmer's Heuristic Algorithm is a method for solving job scheduling problems by assigning weights to machines and evaluating job weights based on processing times. The algorithm's steps involve calculating weights, sorting jobs, forming a sequence, and computing the Make-Span to determine an efficient job schedule.

4.1.1. Time Complexity of Palmer's Heuristic

The complexity for *Step 2* is $O(m)$, for *Step 3*, it is $O(n)$, for *Step 4-5*, it is $O(n\log(n))$, and for *Step 6*, it is $O(mn)$. Hence, the complexity of Palmer's Heuristic = $O(m+n+n\log(n)+mn) = O(mn+n\ \log(n))$.

4.2. CDS Heuristic

The CDS (Campbell, Dudek, and Smith) heuristic algorithm employs an iterative approach that integrates Johnson's algorithm to progressively refine towards an improved optimal solution. The primary objective is to efficiently schedule a set of 'n' jobs on 'm' machines. This method involves several systematic steps that involve the creation of sequences, application of extended Johnson's algorithm, and the determination of the best achievable makespan.

For m machines: $M1, M2, M3 \ldots\ldots M(m)$ and n jobs:

Step 1. Initiate the process by creating a set of $(m-1)$ sequences, as shown in Table 4.

Table 4: The set of $(m-1)$ distinct sequences for machine combination

$\boldsymbol{S1}$	$\boldsymbol{M1}$	$\boldsymbol{M(m)}$
$\boldsymbol{S2}$	$M1 + M2$	$M(m-1) + M(m)$
$\boldsymbol{S3}$	$M1 + M2 + M3$	$M(m-2) + M(m-1) + M(m)$
...	$M1 + M2 + M3 + \cdots$	...
$\boldsymbol{S(m-1)}$	$M1 + M2 + M3 + \cdots + M(m-1)$	$M2 + M3 + \cdots + M(m)$

Step 2. For each of the $(m-1)$ sequences generated in *Step 1*, the extension of Johnson's algorithm is applied. This variant of Johnson's algorithm optimizes the job order within each sequence to minimize the completion times.

Step 3. Among the outcomes obtained in *Step 2*, the sequence yielding the most favorable makespan is selected as the optimal solution.

Step 4. The CDS heuristic assesses a total of $(m-1)$ sequences through the aforementioned process.

In essence, the CDS heuristic algorithm combines the strengths of Johnson's algorithm and iterative sequencing to systematically improve job scheduling on machines. The algorithm sequentially generates and refines sequences, applying a specialized extension of Johnson's algorithm to optimize job orders. Eventually, by evaluating multiple sequences, the algorithm selects an optimal or near-optimal solution to the scheduling problem to minimize the makespan and enhance overall efficiency.

4.2.1 Time Complexity of CDS Heuristic

For *Step 1*, to create the sequence for n jobs, the complexity would be $O\big(n * (m-1) * (m-1)\big)$. The complexity of *Step 2* is $O(m * n \log(n))$.

Hence, the complexity of CDS Heuristic $= O\big(n * (m-1) * (m-1)\big) + O(m * n \log(n)) = O\big(m^2 n + mn \log(n)\big)$.

4.3. Gupta's Heuristic

Gupta's heuristic algorithm presents a systematic approach for solving job scheduling problems. This technique involves several well-defined steps, including function calculations and recursive relations, to optimize the scheduling of 'n' jobs onto 'm' machines. The objective is to minimize the overall processing time and enhance the efficiency of the schedule.

Step 1. Initialize by defining A_j as follows:

$$A_j = \begin{cases} 1 & \text{if } t_{mj} \leq t_{1j}, \\ -1 & \text{otherwise.} \end{cases} \quad (3)$$

Here, A_j is a binary decision variable that is used to determine the priority of job j, t_{mj} represents the processing time of job j on machine m, and t_{1j} represents the processing time of job j on machine 1.

Step 2. Compute the function value $f(j)$ associated with job j using the formula:

$$f(j) = \frac{A_j}{\min(t_{ij} + t_{i+1,j})} \quad \text{for } j = 1, 2, \ldots, n \text{ and } i = 1, 2, \ldots, (m-1) \quad (4)$$

Here, t_{ij} denotes the processing time of job j on machine i, and $t_{i+1,j}$ represents the processing time of job j on machine $(i+1)$. This computation is performed for all j from 1 to n and i from 1 to $(m-1)$.

Step 3. Arrange the 'n' jobs in ascending order based on their calculated $f(j)$ values. This arrangement prioritizes the job with the least total processing time across all m machines.

Step 4. Determine the makespan of the predetermined schedule using the recursive relation:

$$T_{ij}^{k} = \max\left(T_{ij}^{(k-1)}, T_{i-1,j}^{k}\right) + t_{ij} \quad (5)$$

In this equation, T_{ij}^{k} represents the cumulative processing time up to the k-th order for job j on machine i, and $T_{i-1,j}^{k}$ represents the cumulative processing time up to the k-th order for job j on machine $(i-1)$.

In summary, Gupta's heuristic algorithm offers a systematic strategy for addressing job scheduling complexities. By incorporating function calculations and recursive relations, it establishes an optimized job order, eventually striving to minimize total processing time and enhance scheduling efficiency.

4.3.1. Time Complexity of Gupta's Heuristic

The complexity for *Step 1* is $O(1)$. For n jobs, *Step 2* would take $O(mn)$ computations. The complexity of *Step 3* is $O(n \log(n))$, and for *Step 4*, it is $O(mn)$. Hence, the time complexity of Gupta's heuristic $= O(n \log(n) + mn)$.

4.4. RA Heuristic

Dannenberg developed an algorithm known as rapid access (RA) that aims to leverage the strengths of both Palmer's slope index method and CDS's method for job scheduling. The approach involves designing an artificial two-machine problem that emulates the characteristics of Palmer's slope index. Subsequently, Johnson's algorithm is employed to optimize the scheduling process. The steps involved in RA heuristic method are as follows:

Step 1. Start by defining $w_{j1} = m - (j-1)$ and $w_{j2} = j$. These terms are chosen to emulate the behavior of the machines and the jobs in the artificial two-machine problem.

Step 2. Formulate processing times as follows:

$$t'_{i1} = \sum_{j=1}^{m} w_{j1} t_{ij} \quad \text{and} \quad t'_{i2} = \sum_{j=1}^{m} w_{j2} t_{ij} \quad (6)$$

Step 3. Determine sets U and V based on the comparison of the generated processing times:

$$U = \{i | t'_{i1} < t'_{i2}\} \text{ and } V = \{i | t'_{i1} \geq t'_{i2}\} \quad (7)$$

Step 4. Sort jobs in set U in ascending order based on the values of t'_{i1}.

Step 5. Sort jobs in set V in descending order based on the values of t'_{i2}.

Step 6. Schedule the jobs onto the machines in the sorted order of set U, and then proceed to schedule the remaining jobs in the sorted order of set V.

In essence, Dannenberg's RA algorithm combines elements from Palmer's slope index method and CDS's method by creating an artificial two-machine problem and optimizing it using Johnson's algorithm. The algorithm strives to strike a balance between these two methods to achieve an efficient job scheduling outcome. By intelligently sorting and scheduling jobs based on their artificial processing times, the RA algorithm provides a systematic approach to solving complex scheduling challenges.

4.4.1. Time Complexity of RA Heuristic

For *Steps 1-3*, complexity is $O(m)$. The complexity of *Steps 4* and *5* is $O(n \log(n))$. Finally, the complexity for *Step 6* is $O(mn)$. Hence, the time complexity of RA heuristic $= O(m + n \log(n) + mn) = O(n \log(n) + mn)$.

4.5. NEH Heuristic

The heuristic algorithm, devised by Nawaz, Enscore, and Ham (NEH) in 1983, is recognized as the Insertion Algorithm. This approach is aimed at minimizing the makespan, a key metric in job scheduling problems. The NEH algorithm consists of a series of systematic steps designed to optimize the arrangement of jobs on machines. Here are the detailed steps of the NEH algorithm.

Step 1. For each individual job j with processing times t_{ij} on machine i, compute the total processing time T_j as:

$$T_j = \sum_{i=1}^{m} t_{ij} \quad (9)$$

Here, m is the number of machines.

Step 2. Arrange the jobs in descending order based on their processing times for each respective machine.

Step 3. Select the first two jobs from the ordered sequence derived in *Step 2*, generate different combinations from them, and then arrange these combinations in a manner that minimizes the partial makespan.

Step 4. Compute the makespan for each of the generated combinations.

Step 5. Choose the combination that yields the lowest makespan.

Step 6. Integrate the subsequent job from the sequence obtained in *Step 2*.

Step 7. Proceed to explore all possible combinations involving the three jobs currently under consideration.

Step 8. Reiterate the calculations performed in *Step 4*, followed by the selection process in *Step 5*, and then proceed to integrate the job according to *Step 6*.

Step 9. Continue to iterate through this process until all jobs have been incorporated.

In summary, the NEH heuristic algorithm involves iteratively inserting jobs into a sequence in a way that minimizes the makespan. By evaluating different insertion positions and selecting the one that leads to the smallest makespan, the algorithm gradually constructs an optimal or near-optimal solution for job scheduling. This algorithm is widely used due to its simplicity and effectiveness in providing good scheduling solutions.

4.5.1. Time Complexity of NEH Heuristic

The complexity in *Step 1* is $O(n)$. *Step 2* uses sorting; using an efficient sorting algorithm, this complexity can be $O(n\log(n))$. The algorithm iteratively inserts each job into the partial sequence to minimize makespan. The insertion procedure, along with evaluating the makespan for each insertion, takes $O(mn^2)$ time for m machines. We require a possible total of $O(n^2)$ positions to insert the jobs into the sequence. Hence, the total runtime would be $O(mn^4)$. But using a heap data structure, the time complexity can be reduced to $O(mn^3\log(n))$. Hence, the time complexity of the NEH heuristic $= O(mn^3\log(n))$.

4.6. RC Heuristic

Rajendran and Chaudhuri (RC) introduced this method to schedule flow shops with the goal of making jobs finish faster (minimizing the makespan) and reducing the total time (flow-time). Here we explain how this heuristic algorithm works, step by step:

Step 1. Set $p = 1$.

Step 2. Determine: $T_i = \sum_{j=1}^{m}[(m-j+1)t_{ij}]\,, \quad i = 1, 2, \dots, n$

Step 3. Create an array of jobs by organizing them in increasing order based on the value of T_i. If there are ties, resolve them using any method.

Step 4. Consider the array of jobs as the schedule and assess it in terms of the overall flow-time of jobs.

Step 5. If $p = m$, go to *Step 8*.

Step 6. Update: $T_i = T_i - \sum_{j=1}^{p} t_{ij}\,, \quad i = 1, 2, \dots, n$

Step 7. Let $p = p + 1$ and go back to *Step 3*.

Step 8. Select the schedule from the maximum of *m* distinct schedules generated, which reduces the total flow-time to the greatest extent. $STOP$.

In summary, RC heuristic algorithm works by iteratively organizing jobs based on a calculated parameter, evaluating schedules, and selecting the one that minimizes flow-time, ultimately aiming to expedite job completion and reduce total processing time.

4.6.1. Time Complexity of RC Heuristic

For each job, *Step 2* runs m times. Therefore, the computation time is $O(mn)$. The complexity for *Steps 3* to *6* is $O(n\log n)$ for each machine. So, for m machines, the complexity will be $O(mn\log(n))$. Hence, the time complexity of RC heuristic $= O(mn\log(n))$.

4.7. Modrak's Heuristic

The step-by-step algorithm is given as follows.

Step 1. Calculate the total processing time of n jobs on machine M_1. Repeat *Step 1* for machines $j = 1, 2, 3, \dots, m$.

Step 2. Divide the m machines into two groups so that,

$$\sum_{j=1}^{x} T_i \sim \sum_{j=x+1}^{m} T_i \rightarrow Minimum \tag{10}$$

Step 3. Determine the total count of machines in each group. Designate the number of machines in Group I as 'a', and the number of machines in Group II as 'b'.

Step 4. Compute the entire operational time, denoted as T, for the jobs within each group utilizing the provided formula:

a) For the Group I and Job (J_1):

$$T_{J1}^{I} = (a \cdot t_{11}) + [(a-1) \cdot t_{12}] + [(a-2) \cdot t_{13}] + \cdots + (1 \cdot t_{1a}) \tag{11}$$

b) For the Group II and Job (J_1):

$$T_{J1}^{II} = (b \cdot t_{1m}) + \left[(b-1) \cdot t_{1,m-1}\right] + \left[(b-2) \cdot t_{1,m-2}\right] + \cdots + \left(1 \cdot t_{1,a+1}\right) \tag{12}$$

Similarly, calculate these values for jobs $J2, J3, \dots, Jn$.

Step 5. List these values in a table with two rows.

Step 6. Implement the final stage of Johnson's rule to determine the optimal sequence.

Step 7. Compute the makespan duration for the sequence derived in *Step 6*.

In summary, Modrak's Heuristic is a comprehensive algorithm that starts by calculating processing times on different machines, divides machines into two groups to balance workloads, and computes operational times for each job within these groups. The algorithm employs Johnson's rule to find the optimal job sequence, resulting in a minimized makespan duration. This approach is valuable for improving efficiency and reducing job completion times in manufacturing and production settings.

4.7.1. Time Complexity of Modrak's Heuristic

The complexity for *Step 1* is $O(mn)$, for *Steps 2* and *3* is $O(m)$, and for *Steps 4* and *5* is $O(n)$. *Step 6* takes $O(n \log(n))$ computations. Hence, the time complexity of Modrak's heuristic $= O(mn + n\log(n))$.

4.8. Time Deviation Method

The time deviation approach involves creating a time duration table for each job in terms of both rows and columns, with an aim to find the best sequence of jobs.

1. The difference between the maximum time duration of the row and the time duration of the cell provides the row deviation for that cell in the time duration table.

$$p_{ij} = r_i - t_{ij} \tag{13}$$

Where r_i represents the highest time of the i^{th} row, p_{ij} denotes the row time deviation of the $(i,j)^{th}$ cell, and t_{ij} signifies the time needed for processing the i^{th} job on the j^{th} machine.

2. The difference between the maximum time duration of the column and the time duration of the cell provides the column deviation for that cell in the time duration table.

$$c_{ij} = s_i - t_{ij} \tag{14}$$

Here, s_i represents the highest time of the i^{th} column, c_{ij} indicates the column time deviation of the $(i,j)^{th}$ cell, and t_{ij} signifies the time needed for processing the i^{th} job on the j^{th} machine.

4.8.1. Algorithm for Sequencing $\mathbf{n}$ *Jobs and* $\mathbf{m}$ *Machines*

The sequence of N jobs across M machines can be determined through the subsequent steps:

Step 1. The initial M-machine problem is transformed into a two-machine problem when either of the following conditions is met, or when both conditions are satisfied.

$$\text{Min of } M_1 \geq \text{Max of } (M_2, M_3, \dots, M_{m-1}) \tag{15}$$
$$\text{Min of } M_m \geq \text{Max of } (M_2, M_3, \dots, M_{m-1})$$

Step 2. Compute the time deviation table for the modified two-machine sequencing problem.

Step 3. The cell where both time deviation vectors are zero for machine 1 indicates the job that should be executed first.

Step 4. If multiple cells have both vectors as zero, calculate the sum deviation of the respective columns. The cell with the highest sum deviation is prioritized for execution first, and so forth.

Step 5. Similar procedures are applied for machine 2, but in this case, the jobs are scheduled to be executed last.

Step 6. Once more, compute the updated time deviation table for all unallocated jobs and proceed with the aforementioned steps.

Step 7. Halt the process when a sequence involving all jobs is obtained.

Step 8. Reverse the acquired sequence to achieve the minimum total elapsed time.

To summarize, the Time Deviation Method is a sequencing algorithm that operates by creating a time duration table, calculating row and column time deviations, and then following a step-by-step process to determine the optimal job sequence. This method transforms the problem into a two-machine scenario when specific conditions are met, and it prioritizes jobs with zero time deviation vectors, followed by those with the highest sum deviation. The algorithm iteratively refines the sequence until it covers all jobs and ultimately reverses the order to minimize the total elapsed time, offering a practical method to enhance job scheduling across a range of manufacturing and production environments.

4.8.2. Time Complexity of Time Deviation Method

To convert the problem into a two-machine problem, we need $O(mn)$ steps according to *Step 1*. For *n* jobs and *m* machines, the time deviation table calculation will take $O(nm)$ computations in *Step 2*. For *Steps 4* and *5*, complexity is $O(n)$. The reduced time deviation table will now have m rows and $(n-2)$ columns with a dimension of $m*(n-2)$. In the next step, the dimension will be $m*(n-4)$, and so on. Hence, the overall complexity of the Time Deviation Method $= O(mn + m(n-2) + m(n-4) + \cdots + m) = O(mn^2)$.

5. COMPARATIVE ANALYSIS OF SELECTED HEURISTIC ALGORITHMS

To perform the comparative analysis of selected heuristic algorithms, we consider a realistic scenario: a garment factory needs to produce five different styles of shirts (Job 1: Polo Shirt, Job 2: Dress Shirt, Job 3: Henley Shirt, Job 4: Chambray Shirt, and Job 5: Flannel Shirt) on three machines: Cutting (M1), Sewing (M2), and Packing (M3). The processing time (in minutes) for each job on each machine is detailed in the following Table 5.

Table 5: 5-jobs and 3-machines flow-shop scheduling problem

Jobs	Cutting (M1)	Sewing (M2)	Packing (M3)
J1	16	18	12
J2	14	10	11
J3	13	20	15
J4	19	15	19
J5	15	16	16
Total Processing Time	**77**	**79**	**73**

Goodness of Selected Heuristic Algorithms:

The goodness of heuristics measures the error percentage of the heuristics. In order to find out how good the heuristic is, we have to find out the lower bound (LB) for the makespan based on each of the machines. Now, let us first find out the sum of processing times on all three machines.

Lower Bound based on M1, $LB_{M1} = \sum t_{1j} + min\{\sum(t_{2j}, t_{3j})\} = 77 + (10 + 11) = 98$
Lower Bound based on M2, $LB_{M2} = min(t_{1j}) + \sum t_{2j} + min(t_{3j}) = 13 + 79 + 11 = 103$
Lower Bound based on M3, $LB_{M3} = min\{\sum(t_{1j}, t_{2j})\} + \sum t_{3j} = (14 + 10) + 73 = 97$

Now out of the three lower bounds, the maximum one is the best bound.

$$LB = \text{Max } \{LB_{M1}, LB_{M2}, LB_{M3}\} = \text{Max } \{98, 103, 97\} = 103$$

Based on the lower bound, we now definitely know that the makespan cannot be less than 103. So, the makespan can only be 103 or more. Also, we know from here that the optimum makespan can be 103 or more based on the lower bound. Now, optimal makespan by,

1. RC Heuristic = 118
2. Time Deviation Method = 116
3. Modrak's Heuristic = 111
4. Gupta's Heuristic = 108

5. RA Heuristic = 108
6. Palmer's Heuristic = 107
7. CDS Heuristic = 107
8. NEH Heuristic = 106

$$\therefore \text{Goodness of Heuristics} = \frac{\text{Makespan} - \text{Lower Bound}}{\text{Lower Bound}} \times 100\% \quad (16)$$

The performance of all eight heuristics, including their optimal sequences, resulting makespans, and deviation from the lower bound (goodness), is summarized in Table 6.

Table 6: Goodness measurement of the selected heuristic algorithms

Heuristics	***Optimal Sequence***	***Makespan (minutes)***	***Goodness*** $\left(\frac{MS-LB}{LB} \times 100\%\right)$
RC Heuristic	J2-J1-J5-J3-J4	118	$\frac{118-103}{103} \times 100\% \approx 14.56\%$
Time Deviation Method	J2-J1-J4-J3-J5	116	$\frac{116-103}{103} \times 100\% \approx 12.62\%$
Modrak's Heuristic	J4-J5-J3-J1-J2	111	$\frac{111-103}{103} \times 100\% \approx 7.76\%$
Gupta's Heuristic	J5-J3-J4-J1-J2	108	$\frac{108-103}{103} \times 100\% \approx 4.85\%$
RA Heuristic	J5-J3-J4-J1-J2	108	$\frac{108-103}{103} \times 100\% \approx 4.85\%$
Palmer's Heuristic	J3-J5-J4-J2-J1	107	$\frac{107-103}{103} \times 100\% \approx 3.88\%$
CDS Heuristic	J3-J5-J4-J1-J2	107	$\frac{107-103}{103} \times 100\% \approx 3.88\%$
NEH Heuristic	J3-J4-J5-J1-J2	106	$\frac{106-103}{103} \times 100\% \approx 2.91\%$

Among the selected heuristics, the NEH Heuristic emerges as the best performer with a makespan of 106 minutes, achieving a minimal error percentage of 2.91%. This summary indicates that the NEH Heuristic effectively minimizes the completion time of the flow-shop scheduling problem while precisely approximating the optimal job sequence. The following Table 7 presents a comprehensive analysis of the optimal job sequence determined by NEH heuristic.

Table 7: Optimal job sequence, elapsed time and idle time determined by NEH heuristic

Optimal Job Sequence:	J3 → J4 → J5 → J1 → J2
Total Minimum Elapsed Time:	106.00 (minutes)
Idle Time for Machine 1:	29.00 (minutes)
Idle Time for Machine 2:	27.00 (minutes)
Idle Time for Machine 3:	33.00 (minutes)
Total Idle Time:	89.00 (minutes)

The Gantt chart visualizing this optimal schedule (sequence J3 → J4 → J5 → J1 → J2) is presented in Figure 2, providing a clear timeline of job processing and machine utilization.

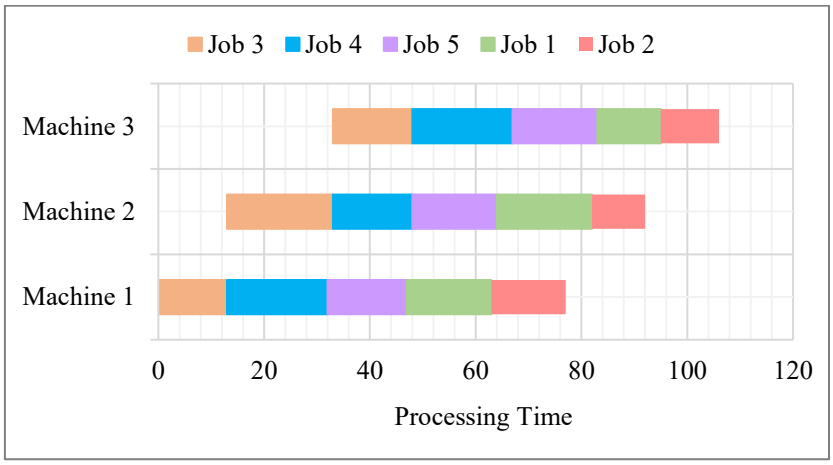

Figure 2: Gantt chart for optimal job sequence (3, 4, 5, 1, 2)

6. FINDINGS AND COMPLEXITY ANALYSIS

6.1. Empirical Results and Insights

With the objective of minimizing the total elapsed time (makespan) and the idle time in flow-shop scheduling problems, the following findings are obtained:

1. When there are three (3) or more machines, if Johnson's method or other exact algorithm fails, the selected heuristic algorithms provide a better optimal solution.
2. The number of generated sequences, time complexity, strength, and weakness of the selected heuristics are presented in Table 8 for an n-job and m-machine flow-shop scheduling problem.

Table 8: The evaluation of heuristics for n jobs and m machines scheduling problems

Heuristics	***Number of Generated Sequences***	***Time Complexity***	***Strength***	***Weakness***
RC Heuristic	1	$O(mn\log(n))$	Simple, easy implementation	Might fail for large-scale scheduling
Time Deviation Method	1	$O(mn^2)$	Minimizes time deviation	Not guaranteed to be optimal
Modrak's Heuristic	1	$O(n\log(n)+mn)$	Effective for small problems, minimizes idle time	Not guaranteed to be optimal
Gupta's Heuristic	1	$O(n\log(n)+mn)$	Effective for small problems	Not guaranteed to be optimal

Table 8: (continued)

Heuristics	*Number of Generated Sequences*	*Time Complexity*	*Strength*	*Weakness*
RA Heuristic	1	$O(n\log(n)+mn)$	Effective for small problems	Not guaranteed to be optimal
Palmer's Heuristic	1	$O(n\log(n)+mn)$	Effective for small problems	Not guaranteed to be optimal
CDS Heuristic	$(m-1)$	$O(m^2n+mn\log(n))$	Effective for small problems, minimizes makespan	Limited to small problem sets
NEH Heuristic	$n!$	$O(mn^3\log(n))$	Mostly produces good solutions, minimizes makespan	Might fail for large scale scheduling

3. NEH heuristic algorithm evaluates more sequences than the other seven heuristic approaches, and therefore the computational time required for this algorithm is much more than the other heuristics.
4. Among the eight heuristic algorithms we have examined, the NEH approach gives the most acceptable optimal sequence, which has minimum elapsed time.
5. It has been observed that the NEH heuristic performs better than other heuristic approaches for scheduling problems involving up to four machines. However, the RA (Rapid Access) heuristic is regarded as preferable for the majority of flow-shop scheduling problems as the number of machines grows.
6. The modified Johnson's algorithm for 'm' machines is also helpful for implementing the CDS algorithm.
7. Heuristic scheduling techniques, which produce significantly satisfactory suboptimal solutions, are becoming more and more common among management when dealing with real-world problems. However, there is plenty of research needs to be done in order to find satisfactory solutions to sequencing problems.
8. Overall, which one of these algorithms has the best complexity will depend on the values of m and n. If the value of m is almost equal to n, then the heuristics of Modrak, Gupta, RA, and Palmer will converge to $O(n^2)$ and hence giving a better time complexity among others.
9. Flow-shop scheduling algorithms often deal with a matrix-like representation where jobs are processed on multiple machines, usually taking the form of an $n \times m$ matrix, where n represents the number of jobs and m represents number of machines, leading to a space complexity of $O(mn)$.
10. Some algorithms may introduce new data structures or optimizations that have an effect on memory requirements. For example, algorithms that use parallel processing or memory-saving strategies may exhibit varying space complexity characteristics. So, while a fundamental understanding of the space complexity of the matrix-like representation is useful, it is critical to remember that true memory requirements can vary based on algorithm-specific details, its implementation and optimization.

6.2. Managerial and Theoretical Implications

Managerial Implications: Production managers in small to medium-sized manufacturing systems (e.g., 3–4 machines and fewer than 20 jobs) can rely on NEH for optimal scheduling performance, especially when solution quality is prioritized over execution time. In contrast, heuristic strategies such as RA or Gupta are better suited for large job shops with limited computational budgets, offering a practical trade-off between runtime efficiency and solution quality.

Theoretical Implications: From a theoretical perspective, the NEH heuristic continuously demonstrates the advantages of progressive job insertion logic in static flow shop settings. At the same time, slope-based priority methods, such as RA, provide a unique balance between deterministic and adaptive scheduling. This suggests potential for future hybridization with learning-based or metaheuristic strategies.

6.3. Scalability and Practical Trade-Offs

Heuristic scalability becomes a key concern as the number of jobs (n) and machines (m) increases. The NEH algorithm, while yielding high-quality results, struggles with scalability due to its factorial sequence generation and cubic time complexity, making it suitable mainly for static environments with fewer jobs and low scheduling frequency.

For high-throughput or real-time environments, heuristics like Modrak, Gupta, RA, and Palmer provide favorable complexity bounds ($O(mn + n\log n)$), making them more computationally viable even if they compromise a few percentage points in makespan performance. Additionally, component-level testing reveals diminishing returns when certain complexity-intensive stages (e.g., iterative reordering in NEH) are retained without proportionate benefit.

Thus, decision-makers should consider job volume, scheduling frequency, and computation time when selecting heuristics. For systems handling over 30 jobs on more than 5 machines, simpler heuristics should be prioritized unless precise scheduling is essential.

7. CONCLUSIONS AND FUTURE RECOMMENDATIONS

In this paper, we have analyzed several heuristic algorithms for solving flow-shop scheduling problems. Finding the optimum sequence with a minimum finish time for flow-shop scheduling is NP-complete which is why these heuristic algorithms are good choices to provide a near-optimal solution quickly. Our numerical results show that the NEH heuristic gives the best result in terms of minimizing makespan and goodness. However, its large time complexity can be impractical for large-scale scheduling. The time deviation method can be a simple and effective solution, although it might be outperformed by other heuristic algorithms in terms of finding the optimal sequence. Overall, the choice of algorithms will depend on the number of jobs and their processing times, the number of machines, and the scheduling objectives. Sometimes, more complex algorithms like metaheuristics might need to be applied to find better solutions.

Abbreviation and Nomenclature

FSP	*Flow-Shop Scheduling Problem*	***SI***	*Slope Index*
RA	*Rapid Access*	***LB***	*Lower Bound*
OS	*Optimal Sequence*	***MS***	*Makespan*
ET	*Elapsed Time*	***IT***	*Idle Time*

Funding: This research received no external funding.

REFERENCES

[1] Michael L. Pinedo, *Scheduling theory, algorithms, and systems*. Springer, 2022.
[2] P. Brucker, *Scheduling algorithms*. Springer Berlin Heidelberg, Germany, 2007.
[3] I. M. Alharkan, *Algorithms for sequencing and scheduling*. Industrial Engineering Department, King Saud University, Riyadh, Saudi Arabia, 2005.
[4] Md. K. A. Asif, S. T. Alam, S. Jahan, and Md. R. Arefin, "An empirical analysis of exact algorithms for solving non-preemptive flow shop scheduling problem," *International Journal of Research in Industrial Engineering*, vol. 11, no. 3, pp. 306–321, Sep. 2022, doi: 10.22105/RIEJ.2022.350120.1324.
[5] R. W. Conway, W. L. Maxwell, and L. W. Miller, *Theory of scheduling*. Addison-Wesley Publishing Company, 1967.
[6] K. R. Baker, *Introduction to sequencing and scheduling*. John Wiley & Sons, Inc., New York, USA, 1974.
[7] S. M. Johnson, "Optimal two- and three-stage production schedules with setup times included," *Naval Research Logistics Quarterly*, vol. 1, no. 1, pp. 61–68, Mar. 1954, doi: 10.1002/NAV.3800010110.
[8] D. S. Palmer, "Sequencing Jobs Through a Multi-Stage Process in the Minimum Total Time—A Quick Method of Obtaining a Near Optimum," *Journal of the Operational Research Society*, vol. 16, no. 1, pp. 101–107, Mar. 1965, doi: 10.1057/JORS.1965.8.
[9] H. G. Campbell, R. A. Dudek, and M. L. Smith, " A Heuristic Algorithm for the n Job, m Machine Sequencing Problem ," *Manage Sci*, vol. 16, no. 10, p. B-630-B-637, Jun. 1970, doi: 10.1287/MNSC.16.10.B630.
[10] J. N. D. Gupta, "A Functional Heuristic Algorithm for the Flowshop Scheduling Problem," *Journal of the Operational Research Society 1971 22:1*, vol. 22, no. 1, pp. 39–47, Mar. 1971, doi: 10.1057/JORS.1971.18.
[11] D. G. Dannenbring, "Evaluation of Flow Shop Sequencing Heuristics," *Manage Sci*, vol. 23, no. 11, pp. 1174–1182, 1977, doi: 10.1287/MNSC.23.11.1174.
[12] M. Nawaz, E. E. Enscore, and I. Ham, "A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem," *Omega (Westport)*, vol. 11, no. 1, pp. 91–95, 1983, doi: 10.1016/0305-0483(83)90088-9.
[13] C. Rajendran and D. Chaudhuri, "An efficient heuristic approach to the scheduling of jobs in a flowshop," *Eur J Oper Res*, vol. 61, no. 3, pp. 318–325, Sep. 1992, doi: 10.1016/0377-2217(92)90361-C.
[14] C. Koulamas, "A new constructive heuristic for the flowshop scheduling problem," *Eur J Oper Res*, vol. 105, no. 1, pp. 66–71, Feb. 1998, doi: 10.1016/S0377-2217(97)00027-1.
[15] V. Modrák and R. S. Pandian, "Flow shop scheduling algorithm to minimize completion time for n-jobs m-machines problem," *Tehnički vjesnik*, vol. 17, no. 3, pp. 273–278, Sep. 2010.
[16] N. R. N, N. R. O, and R. B. I, "Modified Heuristic Time Deviation Technique for Job Sequencing and Computation of Minimum Total Elapsed Time," *International Journal of Computer Science and Information Technology*, vol. 5, no. 3, pp. 67–77, Jun. 2013, doi: 10.5121/IJCSIT.2013.5305.
[17] J. Lin, Z. J. Wang, and X. Li, "A backtracking search hyper-heuristic for the distributed assembly flow-shop scheduling problem," *Swarm Evol Comput*, vol. 36, pp. 124–135, Oct. 2017, doi: 10.1016/J.SWEVO.2017.04.007.
[18] Y. Yang and X. Li, "A knowledge-driven constructive heuristic algorithm for the distributed assembly blocking flow shop scheduling problem," *Expert Syst Appl*, vol. 202, Sep. 2022, doi: 10.1016/J.ESWA.2022.117269.
[19] Z. Pan, D. Lei, and L. Wang, "A Knowledge-Based Two-Population Optimization Algorithm for Distributed Energy-Efficient Parallel Machines Scheduling," *IEEE Trans Cybern*, vol. 52, no. 6, pp. 5051–5063, Jun. 2022, doi: 10.1109/TCYB.2020.3026571.

[20] F. Zhao, S. Di, and L. Wang, "A Hyperheuristic With Q-Learning for the Multiobjective Energy-Efficient Distributed Blocking Flow Shop Scheduling Problem," *IEEE Trans Cybern*, vol. 53, no. 5, pp. 3337–3350, May 2023, doi: 10.1109/TCYB.2022.3192112.

[21] F. Zhao, C. Zhuang, L. Wang, and C. Dong, "An Iterative Greedy Algorithm With *Q* -Learning Mechanism for the Multiobjective Distributed No-Idle Permutation Flowshop Scheduling," *IEEE Trans Syst Man Cybern Syst*, vol. 54, no. 5, pp. 3207–3219, May 2024, doi: 10.1109/TSMC.2024.3358383.

[22] B. Jerbi, S. Kanoun, and H. Kamoun, "Multi-objective mathematical programs to minimize the makespan, the patients' flow time, and doctors' workloads variation using dispatching rules and genetic algorithm," *Yugoslav Journal of Operations Research*, vol. 35, no. 2, pp. 365–382, 2025, doi: 10.2298/YJOR240115032J.