Journal of Robotics and Control (JRC)
Volume 5, Issue 6, 2024
ISSN: 2715-5072, DOI: 10.18196/jrc.v5i6.22208

A Model of Proactive-Reactive Job Shop Scheduling to Tackle Uncertain Events with Greedy Randomized Adaptive Search Procedure

Muhammad Usman Nisar [1*], Anas Ma'ruf [2], Andi Cakravastia [3], Abdul Hakim Halim [4]
[1, 2, 3, 4] Graduate Program of Industrial Engineering and Management, Bandung Institute of Technology, Bandung, Indonesia
Email: [1] 33421701@mahasiswa.itb.ac.id, [2] maruf@itb.ac.id, [3] andi@itb.ac.id, [4] ahakimhalim@itb.ac.id
*Corresponding Author

***Abstract*—Despite substantial research on job shop scheduling (JSS), there is a gap owing to the lack of a unified framework that considers exact, heuristic, and metaheuristic methods for JSS. This study addressed this gap by presenting a comprehensive approach. The study offered following contributions in this regard: analyzed the exact optimization method for benchmarking, investigated a greedy algorithm (G_rA) for faster solutions, and implemented a novel Greedy Randomized Adaptive Search Procedure (GRASP) to achieve high-quality solutions with computational effectiveness. Additionally, this study considered serious dynamic events (SDE) such as new job arrivals (NJA), rush order (RO), machine failures (MF), and scheduled machine maintenance (SMM), as scheduling disruptions and proposed a proactive-reactive rescheduling strategy, with right-shift (RF) and regeneration (Reg) methods using a hybrid (periodic and event-driven) policy to tackle them. Results showed that the exact methods are optimal but computationally intensive, G_rA are faster but suboptimal, and GRASP strike a balance, delivering high-quality solutions with only a 3.43% gap from exact methods while maintaining computational efficiency. Additionally, RF method effectively handled MF, while Reg efficiently integrated NJA, RO, and SMM. Overall, this study offered a comprehensive approach to JSS, enhancing applicability in manufacturing environments.**

***Keywords*—Job Shop Scheduling; Dynamic Events; GRASP; Proactive-Reactive Rescheduling.**

I. INTRODUCTION

Scheduling is a critical decision-making problem in production and management systems [1]. It involves allocating jobs to machines to optimize certain objective function [1], [2]. Scheduling plays an important role in most manufacturing systems [3] and has diverse applications such as job allocation at workstations [4], machines in a workshop [5], and many others, contributing to variety of scheduling models [6]. Categorizing these scheduling models is crucial, as the methods for their resolution depend on the problem's nature [7]. These scheduling models are categorized as: (1) single machine, (2) parallel machines, (3) flow shop, (4) job shop, and (5) open shop.

A comparison in terms of characteristics and complexity between these scheduling models has been provided in Fig. 1, showing that singe and parallel machines models belong to P-class complexity problems, and solvable in polynomial time [8]. In contrast, flow shop and open shop are NP-hard, making them highly challenging as they cannot be solved in polynomial time [9]. On the other hand, job shop scheduling (JSS), is strongly NP-hard, showing even higher complexity.

As compared to other scheduling models, JSS gained massive attention because it has: (1) real world applications [21], (2) broad engineering and social background [22], (3) significant influence on manufacturing efficiency [23], and (4) a great deal of scientific research value, making it a crucial factor in scheduling research.

JSS consists assigning n jobs to m machines to optimize a specific objective function under imposed constraints [10], [11], [12], with each job following a predetermined routing [13], [14], [15]. Due to its strong NP-hard nature [17], [18], [19], JSS has been a challenge for over 50 years [16]. This inherent complexity of JSS makes obtaining an optimal solution challenging even for small-scale instances [20].

To address this inherent complexity of JSS, various techniques has been proposed [5]. These techniques range from exact methods (e.g., dynamic and constraint programming, branch and bound, and branch and cut), to metaheuristic techniques (e.g., genetic algorithms (GA) [24], simulated annealing (SA) [25], ant colony optimization (ACO) [26], and tabu search (TS) [27].

Even though JSS has been widely studied, the majority of JSS literature studied it in static conditions, with suppositions: (1) all relevant data is accessible at time zero [28], (2) job characteristics, such as the release date (r_j), due date (d_j), and processing time (p_{ij}), are known [29], (3) no dynamic event happens during processing, and (4) machines are always available (i.e., never break down or require maintenance) [30]. However, these assumptions are very restrictive in real-world settings [31].

In real-world, production environments are often dynamic, where JSS experiences dynamic events [32], [33], [34]. These dynamic events are happening frequently in JSS, leading to frequent system updates, causing nervousness, deviating the original schedule, and reducing the efficiency of scheduled execution [28], making the previously feasible schedule $-$ infeasible [35]. Thus, handling these dynamic events is of practical importance [6].

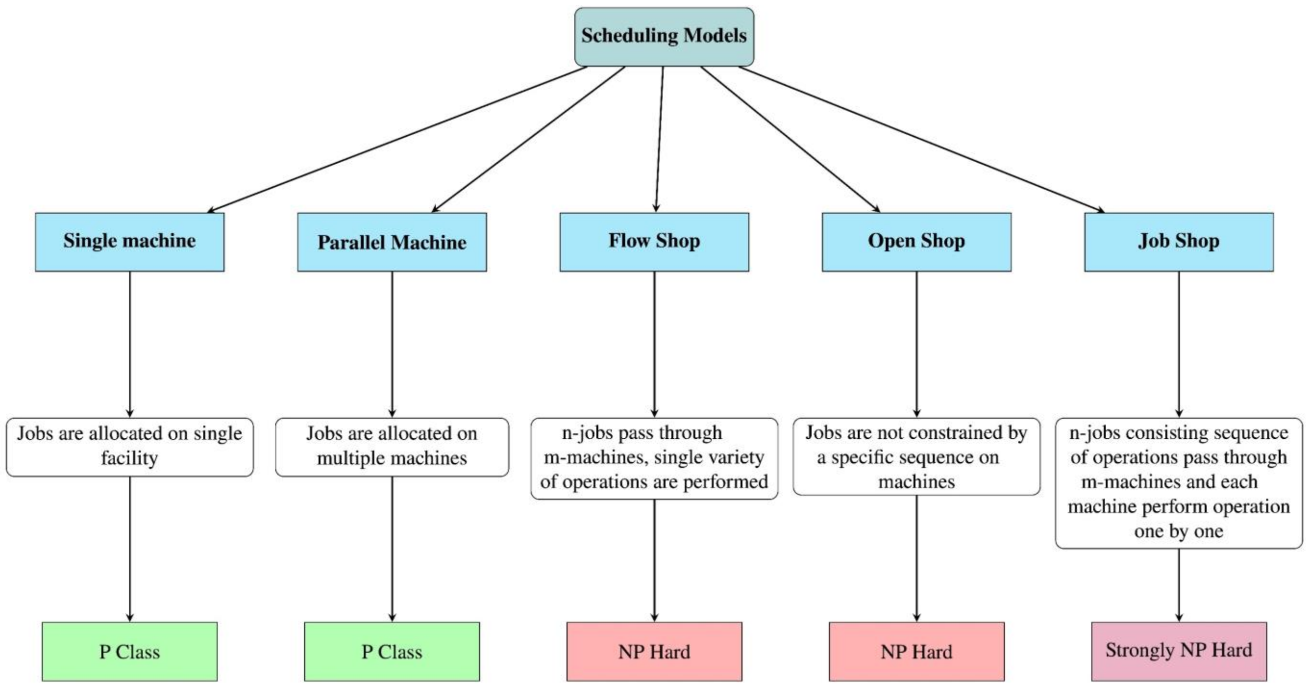
Scheduling Models
Single machine
Parallel Machine
Flow Shop
Open Shop
Job Shop
Jobs are allocated on single facility
Jobs are allocated on multiple machines
n-jobs pass through m-machines, single variety of operations are performed
Jobs are not constrained by a specific sequence on machines
n-jobs consisting sequence of operations pass through m-machines and each machine perform operation one by one
P Class
P Class
NP Hard
NP Hard
Strongly NP Hard

Fig. 1. Scheduling models comparison

Reference [36] classified these dynamic events into serious dynamic events (SDE), medium dynamic events (MDE) and low dynamic events (LDE) based on their impact on the scheduling system. Fig. 2 provides this classification and its implications on scheduling efficiency. SDE impact the scheduling process up to an extent that the schedule becomes invalid and carrying with the same schedule would result in reduced efficiency. MDE, though less intense than SDE, still require measures as they pose challenges to scheduling efficiency by introducing disruptions. LDE, while low individually, can accumulate to affect scheduling efficiency by introducing delays.

Based on the frequency of occurrence, we further categorized SDE into regular serious events (RSE) and significant serious events (SSE). RSE, like new job arrivals (NJA) and scheduled machine maintenance (SMM), are predictable, occurring at regular intervals as part of the normal production cycle. They require job priority adjustments and resource reallocation, respectively, and can be planned for in advance using proactive strategies. In contrast, SSE, such as rush orders (RO) and machine failures (MF), are less predictable and have a higher impact on the production schedule. SSE require immediate attention to prevent substantial disruptions. They cause delays and necessitate reactive rescheduling to minimize downtime.

Numerous authors have studied the impact of SDE on JSS. Reference [37] highlighted the impact of MF on scheduling and role of rescheduling in addressing them. Another study by [38] demonstrated how dynamic rescheduling maintains performance amid RO using reactive strategies. The benefits of dynamic scheduling for NJA have been emphasized by [39]. Additionally, the importance of considering SMM was highlighted by [40]. These studies collectively show that SDE significantly affect scheduling and must be considered. Therefore, addressing SDE through rescheduling is a primary focus of this study.

Rescheduling is a process of updating the previously optimal schedule upon SDE occurrence [8]. Significant literature exists on rescheduling, involving following aspects: rescheduling factor, rescheduling strategies, rescheduling policies, and rescheduling methods [37]. Literature emphasized that rescheduling should address two key questions: “how?” and “when?” [41], [42]. However, to effectively address all aspects of rescheduling, another important question must be considered: "with what?".

1) “How” to reschedule: three strategies are used to address this: completely reactive (R_{act}), proactive (P_{act}), and proactive-reactive ($P_{act} - R_{act}$). R_{act} strategy responds to disruptions based on real-time system information [43]. The drawback is that they act after dynamic events have already impacted the schedule. P_{act} strategy, on the other hand, anticipate potential dynamic events and incorporate them into the initial schedule [44]. However, they do not accurately reflect the current state of the system since they rely on predictions [45]. The best approach is a $P_{act} - R_{act}$ strategy, which combines the strength of both P_{act} and R_{act} by creating an initial schedule using the P_{act} strategy and updating it with R_{act} strategy upon SSE occurrence [46].

2) “When” to reschedule: various policies are used to address this question: periodic (p_e), event-driven (ED), and hybrid (Hyb). A p_e policy reschedules jobs upon RSE occurrence at regular intervals [37], offering stability but potentially compromises performance when SSE occur [7]. The ED policy, on the other hand, effectively handles SSE but they can lead to frequent rescheduling, resulting in high computational costs. In contrast, a Hyb policy is very effective which combines the benefits of both policies (p_e and ED) by rescheduling RSE at fixed intervals through p_e policy and rescheduling SSE using ED policy [36]. This balanced approach, makes it a preferred choice in dynamic environments [41].

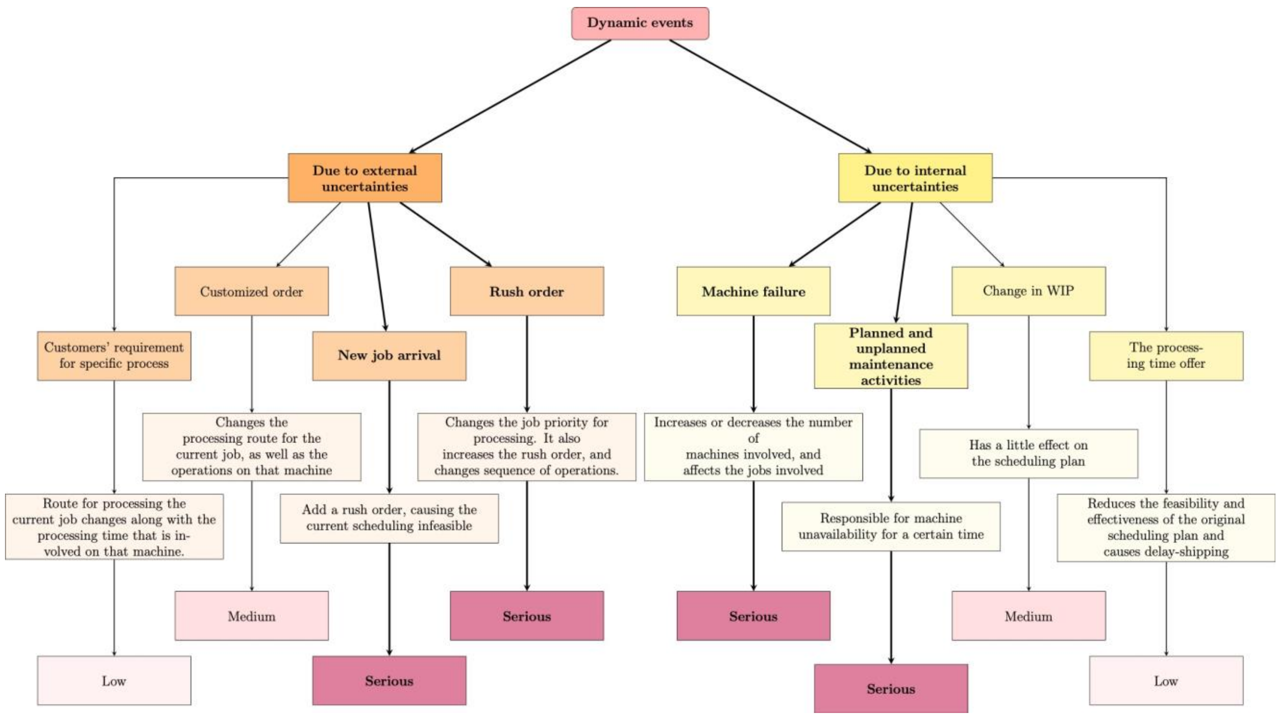
Dynamic events
Due to external uncertainties
Due to internal uncertainties
Customized order
Rush order
Machine failure
Change in WIP
Customers' requirement for specific process
New job arrival
Planned and unplanned maintenance activities
The process-ing time offer
Changes the processing route for the current job, as well as the operations on that machine
Changes the job priority for processing. It also increases the rush order, and changes sequence of operations.
Increases or decreases the number of machines involved, and affects the jobs involved
Has a little effect on the scheduling plan
Route for processing the current job changes along with the processing time that is in-volved on that machine.
Add a rush order, causing the current scheduling infeasible
Responsible for machine unavailability for a certain time
Reduces the feasibility and effectiveness of the original scheduling plan and causes delay-shipping
Medium
Serious
Serious
Medium
Low
Serious
Serious
Low

Fig. 2. Dynamic events and their implications on scheduling

3) "With what" to reschedule: three methods are used to address this question: partial (p_a), right-shift (RF), and regeneration (Reg). A p_a method targets affected operations, balancing stability and solution quality, but struggles with significant events. RF method, a simple and efficient approach, delays remaining operations to maintain feasibility [37], but it may cause delays. Reg method reschedules all unprocessed operations, potentially optimizing the new schedule but at a higher computational cost [7].

Each rescheduling approach offers distinct trade-offs. Our study examined a $P_{act} - R_{act}$ strategy, identified as best approach for handling dynamic events based on previous research by [41]. While [41] examined JSS under events such as NJA and MF with a $P_{act} - R_{act}$ strategy, we expanded on this by considering SDE such as NJA, RO, SMM and MF instead of just two dynamic events and presented RF and Reg methods to reschedule jobs using a Hyb policy. The RF method has the potential to handle MF as it provides a quick solution by delaying the affected operations without significant computational cost. On the other hand, the Reg method effectively integrate NJA, RO, and SMM by re-optimizing the schedule from the point of disruption. This adopted rescheduling approach has the potential to respond to different types of SDE in efficient way.

In general, despite the vast body of research dedicated to addressing JSS problems, a significant gap exists in the literature (shown in Table I). As seen from Table I, existing studies have primarily focused on exploring either the exact optimization [47], heuristic methods [48], or metaheuristic techniques [27], [49] independently, lacking a unified framework that considers these methods. Additionally, most existing studies assume that job attributes, such as release dates, are known in advance [29] and often neglect the occurrence of SDE. These assumptions limit the applicability of solutions in real-world manufacturing, where dynamic events are common. Therefore, a study that unifies through developing and comparing the exact, heuristic, and metaheuristic methods while considering SDE is critically needed to provide more realistic JSS solutions.

This study makes several contributions in this regard: First, it establishes computational constraints and analyzes the complexity of finding optimal schedules using the exact optimization techniques. The optimal solutions obtained through the exact methods are then served as benchmarks for optimality. While the exact method guaranteed optimal solutions, they turned out to be computationally expensive for medium to large instances. To address this, a greedy algorithm (G_rA) has been examined to obtain quick locally optimal solutions. Though G_rA obtained faster solutions, it struggled with optimality. To overcome this, a novel Greedy Randomized Adaptive Search Procedure (GRASP) algorithm is employed, with a more focused and directed procedure for operations swapping considering operations with significant tardiness contribution. This approach balanced exploration and exploitation, improving solution quality. Additionally, this study addresses dynamic JSS scenario involving SDE such as NJA, RO, MF, and SMM. A comprehensive rescheduling procedure using a $P_{act} - R_{act}$ strategy is implemented, featuring RF and Reg methods with a Hyb (p_e and ED) policy. Our research has practicality and real-world relevance in automotive, electronics manufacturing, and other companies.

The rest of the paper is organized as follows: Section II provides a literature review, Section III defines the JSS problem and develops mathematical model; Section IV presents approaches to solve JSS including the exact method, G_rA and GRASP; Section V provides experimental setup, Section VI presents the rescheduling procedure; Section VII provides the results and discussion, followed by the study conclusion and future work.

TABLE I. LITERATURE STUDY ON JSS SHOWING THE POTENTIAL GAP IN STUDIES

References	Exact	Heuristic	GRASP	Serious Dynamic events (SDE)				Strategy	Policy	Method
				NJA	RO	MF	SMM	$P_{act} - R_{act}$	Hyb	$RF\&Reg$
[47], [48], [57], [77]	✓									
[24], [25], [26], [27], [36], [49], [58], [59], [60], [63], [78], [79], [80], [81], [82]		✓								
[68]			✓							
[83], [84]		✓		✓				✓		
[41], [85], [86], [87], [88]		✓		✓		✓		✓	✓	
This study	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

II. LITERATURE REVIEW

A. Related to JSS

JSS has attracted significant attention from scholars, which has resulted in the development of a wide range of approaches to address its complexity. From the exact methods like branch-and-bound [57] and mathematical programming [47] to heuristic approaches like shifting bottleneck [48], and dispatching rules [58], [59], as well as to metaheuristic approaches such as TS [49], GA [60], SA [61], [62], and ACO [63], a plethora of solutions have been proposed by different authors. The GRASP, on the other hand, is a promising approach for solving JSS problems as it efficiently explores the larger search space by combining local search strategies with both greedy and random phases.

Compared to other metaheuristic approaches, GRASP is competitive in solution quality and computational efficiency and is easier to implement and tune [50]: requiring only a few parameters to tune, such as the Restricted Candidate List (RCL) and the number of iterations [51]. [52] demonstrated GRASP's consistent performance in delivering high-quality solutions across various combinatorial optimization problems. Over the past decade, GRASP has been effectively applied to various combinatorial problems [53], [54], [55], [56]. Overall, GRASP is especially well-suited for handling JSS problems because of its adaptability and flexibility.

B. Related to JSS and GRASP

The literature provides a considerable body of research proving the GRASP's successful application in various sectors. These domains include production and manufacturing systems (including discrete manufacturing parts [64], flowshop [65], [66], just-in-time scheduling [67], JSS [68], flexible job shop scheduling (FJSS) [51], [69], [70], [71], and industrial line balancing [72], etc.), routing and logistics (which involves mixed Chinese postman problem [54], traveling salesman problem [73], [74], and vehicle scheduling problem [75][76].

Among the existing research, the study by [68] stands out. They investigated the GRASP approach, which combines greedy and randomized components, by constructing an RCL based on a threshold value and selecting elements probabilistically from the RCL. Their method assigns equal probability to each option in the RCL, emphasizing randomness in the selection process.

In contrast, our proposed methodology takes a more guided approach. We utilize problem-specific information, on the initial solution generated in the construction stage, through tardiness-based local search procedure considering operations with a significant tardiness contribution value, to enhance the quality of the local search. Additionally, random operation swaps and a restart mechanism have been used to extend the search procedure and prevent convergence to local optima, resulting in high-quality solutions with computational effectiveness.

C. Related to JSS and Rescheduling

Rescheduling is a process of updating the previously optimal schedule upon SDE occurrence [8]. Rescheduling is crucial in uncertain manufacturing environments [89], [90]. It minimizes tardiness penalties, and ensure timely order delivery, thereby enhancing efficiency and responsiveness. Various authors have contributed to the existing literature on rescheduling. [37] provided a comprehensive framework for understanding rescheduling research, while [91] conducted a literature review on executing production schedules in the face of unexpected disruptions. Additionally, [3] surveyed dispatching rules for dynamic environments, and [38] explored dynamic scheduling in manufacturing systems. Current research on rescheduling focuses on following aspects: Rescheduling factor, rescheduling strategies, rescheduling policies, and rescheduling methods [37].

1) Rescheduling factors: Addressed single-machine scheduling with NJA for tardiness related objective function [43]. Proposed a technique for dynamic JSS with random NJA and MF [41]. Managed MF in JSS to minimize tardiness [92]. Considered $NJA,$ aiming to minimize weighted tardiness [93]. Analyzed rescheduling under NJA, evaluating schedule stability and efficiency using total completion time and weighted tardiness [94].

2) Rescheduling strategies: various authors have categorized it differently. For example, [37] and [95] categorized four types of rescheduling strategies: $R_{act}, predictive\ (P_{pact}) - R_{act}, P_{act} - R_{act}$ and robust P_{act}. Similarly, [96] covered four types of rescheduling strategies: completely $R_{act}, P_{pact} - R_{act}$, robustness-based, and knowledge-based. Presented a hybrid method that combines $P_{act}\ and\ R_{act}$ strategies [97]. In [83] studied $P_{pact} - R_{act}$ strategy along with P_{act} strategy. In [98] gave an overview of completely R_{act}, robust, and pre-R_{act} strategies. In [93] introduced a $P_{pact} - R_{act}$ strategy for manufacturing control systems, focusing on NJA.

3) Rescheduling policies: In [99] outlined the rescheduling policies (p_e, ED, Hyb). In [100] compared $p_e\ and\ ED$ rescheduling policies for NJA. In [101] introduced analytical methods for p_e *and* ED policies. In [85] presented a p_e *and* ED rolling horizon policy for DJSS. In [42] defined a Hyb policy for DJSS to manage process delay and RO. In [41] proposed a rescheduling policy for solving DJSS with NJA *and* MF. In [36] proposed a Hyb policy to address frequent changes.

4) Rescheduling methods: they are divided into three types: (1) P_a [43], [46], [85], [102], [103], (2) RF [36], [99], [104], and (3) Reg [42], [97], [105]. Reference [106] demonstrated different ways to schedule jobs in a machine for rescheduling process—scheduling new jobs after completing current ones, immediately scheduling new jobs, and inserting new jobs into idle time during scheduling.

5) Rescheduling job scheme: Along with other rescheduling aspects, another important aspect is rescheduling jobs scheme. Various authors have categorized rescheduling jobs scheme differently. Categorized it as jobs not processing yet and in-process jobs [92]. Categorized them as jobs waiting for processing and jobs waiting to be scheduled [42]. Divided them into available jobs, jobs in the jobs window, and already finished jobs [85].

III. Problem Formulation

JSS consists of a set of machines as denoted by $M = \{M_1, M_2 \dots, M_m\}$ and a corresponding set of jobs as represented by $J = \{J_1, J_2, .., J_n\}$. Each job comprises a series of operations, $O_j = \{O_{j1}, O_{j2}, .., O_{jl}\}$. These operations must be processed following a predetermined technological order, which cannot be changed. Each job is assigned a release date (r_j), and a due date (d_j) and each job is allocated on a machine in M to be processed with a given uninterrupted processing time (p_{ij}). Each machine can process one job at a time, and each job can be processed only once on a given machine. A job revisit over the same machine is not allowed, and processing of a job must not be interrupted. All the jobs must be scheduled on machines such that the precedence constraint among different operations of different jobs on the same machine and the dependency constraint among different operations of the same job, must be observed. The objective is to minimize the total tardiness of all jobs.

A flowchart has been presented, as seen in Fig. 3, outlining the approach adopted in this study. The flowchart begins with defining the JSS problem. It then details the development of the exact methods, greedy algorithm (G_rA), and GRASP. Finally, it includes conducting a case study, experimental setup, rescheduling procedure, results and discussions, followed by summarizing conclusion and future research.

D. Mathematical Notions

Sets:

Set of machines	$M = \{M_1, M_2, \dots, M_m\}$
Set of jobs	$J = \{J_1, J_2, .., J_n\}$
Set of operations	$O = \{O_1, O_2, \dots, O_l\}$
Set of rescheduling periods	$R = \{R_1, R_2, .., R_q\}$
Set of unfinished jobs	$J^u = J_1^u, J_2^u, \dots, J_{n^u}^u$
Set of new jobs	$J^n = J_1^n, J_2^n, \dots, J_{n^n}^n$

Index:

Machine index	$i = \{1,2, .., m\}$
Jobs index	$j = \{1,2, .., n\}$
Previous machine index	$i' = \{2,3, .., m\}$
Operation index	$k = \{1,2, \dots, l\}$
Last operation of job index	j_m
Last machine index	i_m
Rescheduling index	$q = \{1,2, .., r\}$
Unfinished jobs index	j'
New jobs index	j''

Parameters:

Number of machines	m_i
Number of jobs	n_j
Release date	r_j
Processing time	p_{ij}
Due date	d_j
Rescheduling point	R_q
Rescheduling interval	$T_q = [R_{q-1}, R_q]$
Number of unfinished jobs	$n'_{j'}$
Number of new jobs	$n''_{j''}$
Last operation on a machine	j_m
Penalty value	V

Variables:

Start time	S_{ij}
Finish time	C_{ij}
Binary variable for jobs assignment on machine i	X_{ijk}
Availability status for machine at rescheduling point	$ASM_i^{R_q}$
Availability status for jobs at rescheduling point	$ASM_j^{R_q}$
Availability time for machine	$ATM_i^{R_q}$
Availability time for job	$ATM_j^{R_q}$

Performance measure:

Total tardiness of all jobs	$\sum_{j \epsilon J} T_j$

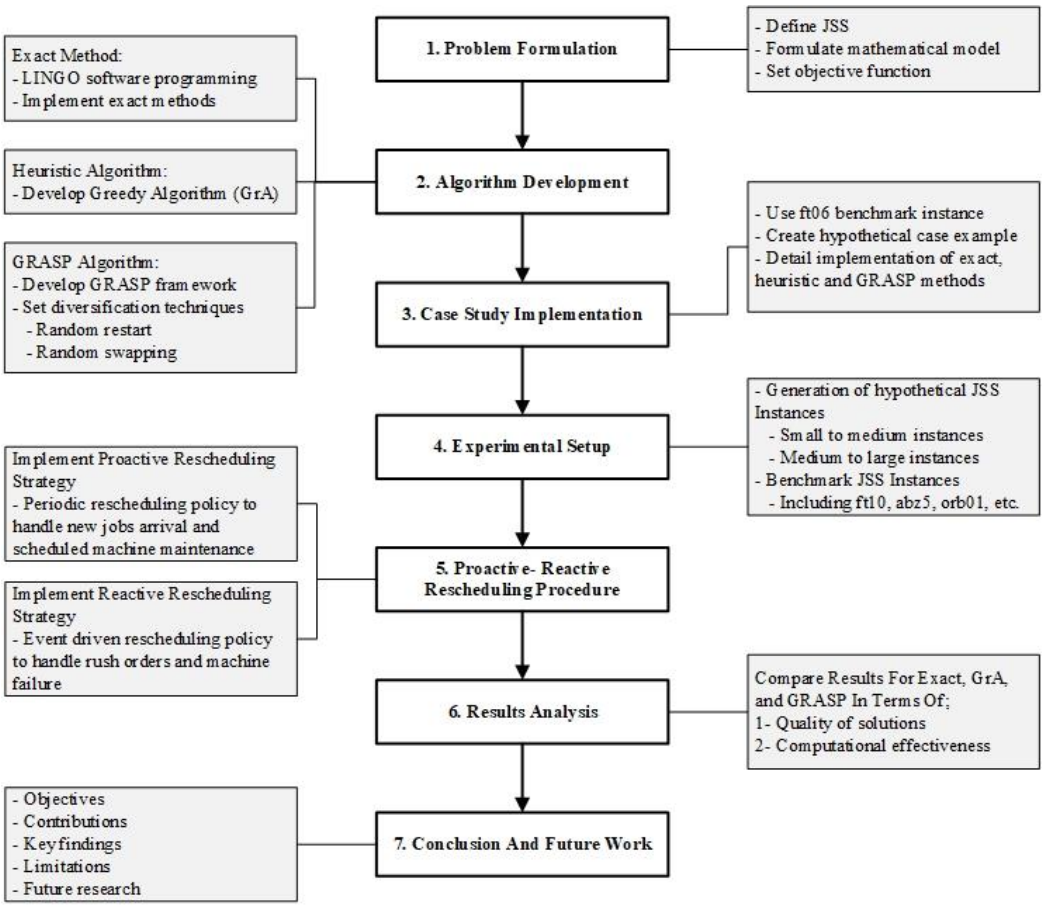
1. Problem Formulation
- Define JSS
- Formulate mathematical model
- Set objective function
2. Algorithm Development
Exact Method:
- LINGO software programming
- Implement exact methods
Heuristic Algorithm:
- Develop Greedy Algorithm (GrA)
GRASP Algorithm:
- Develop GRASP framework
- Set diversification techniques
- Random restart
- Random swapping
3. Case Study Implementation
- Use ft06 benchmark instance
- Create hypothetical case example
- Detail implementation of exact, heuristic and GRASP methods
4. Experimental Setup
- Generation of hypothetical JSS Instances
- Small to medium instances
- Medium to large instances
- Benchmark JSS Instances
- Including ft10, abz5, orb01, etc.
5. Proactive- Reactive Rescheduling Procedure
Implement Proactive Rescheduling Strategy
- Periodic rescheduling policy to handle new jobs arrival and scheduled machine maintenance
Implement Reactive Rescheduling Strategy
- Event driven rescheduling policy to handle rush orders and machine failure
6. Results Analysis
Compare Results For Exact, GrA, and GRASP In Terms Of;
1- Quality of solutions
2- Computational effectiveness
7. Conclusion And Future Work
- Objectives
- Contributions
- Key findings
- Limitations
- Future research

Fig. 3. Research methodology flowchart

E. Mathematical Model

$$\text{Minimize } \left(\sum_{j\epsilon J} T_j\right) \tag{1}$$

Subject to:

$$S_{ij} \geq 0, \forall\ \mathrm{i} \in \mathrm{M}, \mathrm{j} \in \mathrm{J} \tag{2}$$

$$S_{ij_1} \geq r_j, \forall\ i \tag{3}$$

$$S_{i'j} + p_{i'j} \leq S_{ij}, \forall\ \mathrm{i}', \mathrm{i} \in M, \mathrm{j} \in \mathrm{J} \tag{4}$$

$$S_{ij} + p_{ij} \geq S_{ik} + V\left(1 - X_{ijk}\right), \forall\ i \in M\ \delta\ j,k\ \in J, j \neq k \tag{5}$$

$$X_{ijk} + X_{ikj} = 1, \forall\ i \in M\ \delta\ j,k\ \in J, j \neq k \tag{6}$$

$$\sum_{i \in M} X_{ijk} \leq 1, \forall\ i \in M\ \delta\ j,k\ \in J, j \neq k \tag{7}$$

$$C_{ij} = S_{ij} + p_{ij}, \forall\ j \tag{8}$$

$$T_j = \max\left(0, C_{ij_f} - d_j\right), \forall\ j \tag{9}$$

Equation (1) shows the objective function which is to minimize the total tardiness. Tardiness is the delay beyond a job's due date. We aim to complete jobs as close to their due dates as possible, reducing delays and improving efficiency. Constraint (2) ensures that jobs cannot start before time zero. Constraint (3) ensures that each job can only start processing after its release date. This is crucial for modeling realistic scenarios. Constraint (4) provides that dependency relationship among different operations of the same job is respected. This constraint ensures that operations are executed in the correct order. Constraint (5) ensures that the precedence relationship among operations of different jobs on the same machine is respected. This is important for maintaining the correct sequence of operations when multiple jobs share the same machine. Constraint (6) assures that operations from different jobs cannot be processed concurrently on the same machine. This reflects the practical limitation that a machine can only process one operation at a time. Constraint (7) guarantees that the machine processes at most one job at a time. This constraint is essential to avoid overlaps and conflicts in job assignments to machines. Constraint (8) computes the completion time for each job, while constraint (9) calculates the each job's tardiness.

To make the model more realistic, some assumptions in the defined model will be relaxed, bringing it closer to real-world scenario. This includes the introduction of SDE like NJA, RO, SMM, and MF, and the introduction of rescheduling procedure to tackle them. To incorporate these assumptions into the model, following constraints are added.

At the rescheduling point, the machines occupied in processing other jobs, marked as "busy", and symbolized as B_m. While machines not engaged in processing any other job are labelled as "available" and represented as A_m. Similarly, at the rescheduling point, certain jobs may be processed on other machines, marked as "busy" and symbolized with B_j. Conversely, jobs not processing on any other machines are labelled as "available" and represented by the symbol A_j.

The mathematical formulation is given below for checking the rescheduling point, availability status and availability time for machines and jobs.

$$R_q = \begin{cases} 1, & if\ rescheduling\ point\ arrived\ yet \\ 0, & Otherwise \end{cases} \tag{10}$$

$$ASM_i^{R_q} = \begin{cases} A_m, & if\ machine\ is\ not\ processing\ any\ job, \\ B_m, & otherwise \end{cases} \tag{11}$$

$$ASM_j^{R_q} = \begin{cases} A_j, & if\ job\ is\ not\ processing\ on\ any\ machine, \\ B_j, & otherwise \end{cases} \tag{12}$$

$$ATM_i^{R_q} \geq C_{ij_m}, \forall j_m \tag{13}$$

$$ATM_j^{R_q} \geq C_{ij_m}, \forall i \tag{14}$$

Constraint (10) verifies whether the rescheduling point arrived yet. This ensures that rescheduling decisions for periodic rescheduling are made at the rescheduling points. Constraints (11) and (12) examine the availability status of machine i and job j respectively, at a given point, determining whether the machine and job are available for processing or currently busy. Constraints (13) and (14) compute the machine availability time and job availability time for each machine and each job respectively.

To compute the new start times for each job at R_q, the following mathematical formulation is given as follows.

$$S_{ij^u} \geq max(R_q, ATM_i^{R_q}), \forall\, i \in M, j^u \in J', q \in R \tag{15}$$

$$S_{ij^n} \geq max(R_q, ATM_i^{R_q}), \forall\, i \in M, j^n \in J^n, q \in R \tag{16}$$

$$S_{ij^u} + p_{ij^u} \geq S_{ik^u} + V(1 - X_{ij^u k^u}), \forall j^u, k^u \in J^u, j^u \neq k^u, i \in M \tag{17}$$

$$S_{ij^n} + p_{ij^n} \geq S_{ik^n} + V(1 - X_{ij^n k^n}), \forall j^n, k^n \in J^n, j^n \neq k^n, i \in M \tag{18}$$

Constraints (15) and (16) specify the start times of all outstanding operations of old jobs that were still in-process when rescheduling point arrived and new jobs that arrived before the rescheduling point, respectively. Constraints (17) and (18) compute the start times of sequential operations of unfinished operations and newly arriving jobs, respectively. In the next section, the solution methods such as the exact method, G_rA and GRASP are presented to solve the developed JSS problem.

IV. Approaches to Address JSS

To solve the developed JSS problem, our study employs the ft06 JSS instance from [107] as a case example. The ft06, having six machines and six jobs with six operations each, serves as a challenging benchmark due to its complexity, especially given that JSS with more than two machines show exponential growth in alternative schedules [108]. By applying our proposed methodologies to ft06, we aim to demonstrate a step-by-step procedure for solving JSS and validate our approach's performance.

For this purpose, a hypothetical manufacturing company scenario has been generated and named as "hypothetical ft06 bike manufacturing company". The hypothetical ft06 bike manufacturing company produces 6 different types of bikes (jobs) namely mountain bike (J_1), road bike (J_2), hybrid bike (J_3), cruiser bike (J_4), electric bike (J_5), and folding bike (J_6).

Each bike requires 6 operations, including frame cutting and shaping, frame welding, painting the frame, assembling, inspection, and packaging. Each bike processes its operations on 6 different available machines with a predefined routing: cutting and shaping machine (M_1), welding machine (M_2), painting machine (M_3), CNC machine (M_4), assembly machine (M_5), and inspection machine (M_6). After they are done with their processes on all the machines, they are dispatched for shipping.

The routing at which each job will visit each machine is shown in Table II and the parameters such as processing time (p_{ij}), release date (r_j), and due date (d_j) are known and given in Table III. From Table II and Table III, each job follows its specific sequence of operations across different machines, with varying processing times for each operation. For example, J_1 begins its first operation on M_3, requiring 1 time unit. After finishing on M_3, J_1 proceeds to M_1 for its second operation, requiring 3 times unit, and so on. This pattern persists for other operations of J_1 and for other jobs. The generated data in Table II and Table III will be used to test the exact method, G_rA, and GRASP in following sections.

TABLE II. Routing for a FT06 Hypothetical Manufacturing

Jobs	Routing
J1	$M_3(O_1) - M_1(O_2) - M_2(O_3) - M_4(O_4) - M_6(O_5) - M_5(O_6)$
J2	$M_2(O_1) - M_3(O_2) - M_5(O_3) - M_6(O_4) - M_1(O_5) - M_4(O_6)$
J3	$M_3(O_1) - M_4(O_2) - M_6(O_3) - M_1(O_4) - M_2(O_5) - M_5(O_6)$
J4	$M_2(O_1) - M_1(O_2) - M_3(O_3) - M_4(O_4) - M_5(O_5) - M_6(O_6)$
J5	$M_3(O_1) - M_2(O_2) - M_5(O_3) - M_6(O_4) - M_1(O_5) - M_4(O_6)$
J6	$M_2(O_1) - M_4(O_2) - M_6(O_3) - M_1(O_4) - M_5(O_5) - M_3(O_6)$

TABLE III. Parameters for FT06 Hypothetical Manufacturing

Processing Time	**Moutain bike (J_1)**	**Road bike (J_2)**	**Hybrid bike (J_3)**	**Cruiser bike (J_4)**	**Electric bike (J_5)**	**Folding bike (J_6)**
Cutting machine (M_1)	3	10	9	5	3	10
Welding machine (M_2)	6	8	1	5	3	3
Painting machine (M_3)	1	5	5	5	9	1
CNC machine (M_4)	7	4	4	3	1	3
Assembly machine (M_5)	6	10	7	8	5	4
Inspection machine (M_6)	3	10	8	9	4	9
Duedate (d_j)	72	31	56	61	52	72

F. Exact Method

The exact methods are optimization methods that guarantee finding the globally optimal solution to a problem [109]. The exact methods include branch-and-bound [57], mathematical programming [47], and many other. The rationale behind using the exact method is: (1) to define the scope of the study, (2) to ensure that the found solution is the best possible one [110], and (3) benchmarking for evaluating the performance of other approximate methods.

The exact method is employed on hypothetical ft06 manufacturing scenario to find the optimal solution. The exact method identifies the sequence of operations for each bike (job) on each machine to minimize the total tardiness. The resulting optimal solution serves as a benchmark for evaluating G_rA and GRASP. By comparing the solutions obtained through G_rA and GRASP with those obtained from the exact method, we evaluate the performance and effectiveness of these approaches.

To implement the exact method, a programming software has been employed on the ExpertBook, equipped with a 64-bit operating system and an 11th generation Intel(R) Core(TM) i7-1165G7 @ 2.80GHz CPU using 8.00 GB of RAM. The process starts with formulating the problem as a mixed-integer non-linear programming (MINLP) model, incorporating all relevant constraints. The input data, including p_{ij}, r_j, d_j and job routing, is prepared and fed into the programming software. The programming software then applies the exact algorithm to exhaustively explore the solution space and find the optimal solution. Upon completion, the optimization results are recorded, including start times (S_{ij}), finish times (C_{ij}), computational times, job tardiness and total tardiness.

The Table IV presents S_{ij} and C_{ij} for hypothetical ft06 bike manufacturing company scenario. Based on these results, a Gantt chart is drawn in Fig. 4 to visualize and verify the solutions' correctness, particularly in terms of precedence relationship among operations. In Fig. 4, each bar represents an operation, with the values indicating its S_{ij} and C_{ij} on that specific machine. For example, J_3 's first operation on machine M_3 is shown as $\binom{0}{5}$ indicating that J_1 starts at 0 times unit and finishes at 5 times unit.

```
1  Input: $\Omega$ , $\Phi$ , $h$
2  Initialization: $s = \{\}$ //Initialize an empty solution set.
3  while: $\exists (J,O) \in \Omega \; s.t. (J,O) \notin s$ //Include all operations.
4      | $\forall (J,O) \in \Omega$
5      |     | compute: $f(s \cup \{(J,0)\}) = h(s,\{(J,O)\})$
6      | find: $(J,0) = argmin_{(J,O)_i \in \Omega \setminus z} \, h(s,\{J,O\})$
7      | $s = s \cup \{J,O\}$ //Add the selected operations to s-set.
8  Return $s$
```

Fig. 4. Pseudocode for greedy algorithm

The exact method, while providing optimal solutions, becomes computationally costly as problem size increases, limiting their practicality for even moderately sized instances [110] [109]. As a result, applications of the exact method become limited by computational demands. Furthermore, the exact methods are not well-suited to handle dynamic events effectively, as they require re-solving the entire problem when SDE occur, which can be computationally expensive or even infeasible. These reasons necessitate the search for other heuristic approaches. Consequently, our research implements G_rA as an alternative method to address these challenges.

G. Greedy Algorithm

Greedy algorithm (G_rA) is a heuristic approach that uses a local heuristic (h) to build candidate solutions to optimization problems step by step. It begins with an empty solution and keeps adding a finite set of elements to the current partial solution [109], [111]. This process iterates until a complete candidate solution is achieved.

The rationale for implementing G_rA is threefold: (1) It offers a trade-off between computational efficiency and scalability [112], (2) it provides approximate solutions quickly, even for large scale problems, enhancing computational efficiency, and (3) It can be adapted to incorporate the dynamic events.

A finite set of elements (Ω), partial solution (s_p), search space (F), and objective function ((Φ) are some of G_rA's key elements. A finite set of elements (Ω) represents the set of all operations that need to be scheduled.

$$\Omega = \{(J,O) | J = J_1, J_2, .., J_n \; \& \; O = O_1, O_2, \dots, O_l\} \tag{19}$$

The partial solution (s_p) represents a subset of operations that have been scheduled so far.

$$s_p \subset \Omega \tag{20}$$

Search space (F) consists of all possible partial solutions that can be generated. It represents all possible combinations of operations that can be scheduled.

$$\mathrm{F} \subset 2^{\Omega} \tag{21}$$

The objective function (Φ) maps a partial solution to a real number representing the objective value of the solution.

$$\Phi : 2^{\Omega} \rightarrow \mathbb{R} \tag{22}$$

A detailed pseudocode has been given for G_rA in Fig. 5. The G_rA takes Ω (finite set of elements), Φ (objective function), and h (local heuristic) as inputs (line 1). G_rA initializes an empty solution set s (line 2) and continues until all operations are included in the solution (line 3). For each operation (J, O) not yet in the solution (line 4): it computes the value of adding this operation to the current solution using the local heuristic h (line 5). It selects the operation that minimizes total tardiness (line 6). The selected operation is added to the solution set s (line 7). The algorithm returns the completed solution s (line 8).

The G_rA is applied to hypothetical ft06 bike manufacturing company scenario. The G_rA processes data about machines (cutting, welding, painting, assembling, inspection, packaging) and bikes (mountain, road, hybrid, cruiser, electric, folding). Starting with an empty solution, it uses the heuristic to incrementally build a solution by adding operations to the partial solution. Feasible solutions are updated as operations are added. The process repeats until a complete, feasible sequence of operations is obtained.

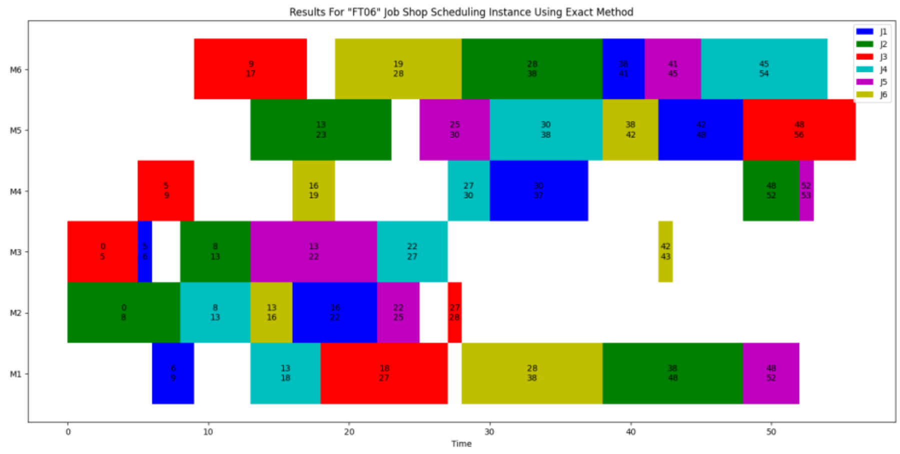

Fig. 5. Gantt chart for ft06 hypothetical manufacturing using exact method

TABLE IV. RESULTS OBTAINED THROUGH EXACT METHOD FOR FT06 HYPOTHETICAL MANUFACTURING

(S_{ij}, C_{ij})	Frame cutting and shaping (O_1)	Frame welding (O_2)	Painting the frame (O_3)	Assembling (O_4)	Inspection (O_5)	Packaging (O_6)	Tardiness (units)
Moutain bike (J_1)	(5,6)	(6,9)	(16,22)	(30,37)	(38,41)	(42,48)	0
Road bike (J_2)	(0,8)	(8,13)	(13,23)	(28,38)	(38,48)	(48,52)	21
Hybrid bike (J_3)	(0,5)	(5,9)	(9,17)	(18,27)	(27,28)	(48,56)	0
Cruiser bike (J_4)	(8,13)	(13,18)	(22,27)	(27,30)	(30,38)	(45,54)	0
Electric bike (J_5)	(13,22)	(22,25)	(25,30)	(41,45)	(48,52)	(52,53)	11
Folding bike (J_6)	(13,16)	(16,19)	(19,28)	(28,38)	(38,42)	(42,43)	0
Computational time							2.11 (sec)

The Table V presents the results obtained, after applying G_rA, for hypothetical ft06 bike manufacturing company scenario in terms of S_{ij}, C_{ij}, tardiness and computational time. Notably, the tardiness value increased for G_rA compared to the exact method, with J_2, J_3 and J_5 showing an increase in tardiness from 21, 0, and 11 to 23, 4 and 13, respectively. The final feasible solution is presented as Gantt chart in Fig. 6. Each bar on Gantt chart represents an operation, with the values indicating its S_{ij} and C_{ij} for each machine.

TABLE V. INITIAL SOLUTION FOR FT06 IN CONSTRUCTION PHASE OF GRASP

Machine	Complete solution
Cutting machine (M_1)	$J_1(O_2)$,$J_4(O_2)$,$J_3(O_4)$,$J_6(O_4)$, $J_2(O_5)$,$J_5(O_5)$
Welding machine (M_2)	$J_2(O_1)$,$J_4(O_1)$,$J_6(O_1)$,$J_1(O_3)$, $J_5(O_2)$,$J_3(O_4)$
Painting machine (M_3)	$J_3(O_1)$,$J_1(O_1)$,$J_2(O_1)$,$J_5(O_1)$, $J_4(O_3)$,$J_6(O_6)$
CNC machine (M_4)	$J_3(O_2)$,$J_6(O_2)$,$J_4(O_4)$,$J_1(O_4)$, $J_2(O_6)$,$J_5(O_6)$
Assembly machine (M_5)	$J_2(O_3)$,$J_5(O_3)$,$J_4(O_5)$,$J_6(O_5)$, $J_1(O_6)$,$J_3(O_6)$
Inspect. machine (M_6)	$J_3(O_3)$,$J_6(O_3)$,$J_2(O_4)$,$J_1(O_5)$, $J_5(O_4)$,$J_4(O_6)$

Results showed that while G_rA offers significant computational efficiency compared to the exact method, it cannot guarantee global optimal solutions [109], [113]. This limitation arises because the G_rA minimizes computational time by making locally optimal decisions without exhaustive searches, resulting in the G_rA getting stuck in local optima.

Consequently, the G_rA struggles to explore the larger solution space, leading to suboptimal solution. Additionally, G_rA generates only a single solution, which is most likely suboptimal [111], and an incorrect decision in early stage may lead to poor solutions in the end [111]. These limitations highlight the need for more advanced optimization techniques. To address these limitations, we proposed a novel GRASP algorithm with a more directed operation swapping procedure aiming to enhance solution quality while maintaining computational efficiency.

H. Greedy Randomized Adaptive Search Procedure

GRASP is an iterative method that effectively solves combinatorial optimization problems [109], striking a balance between greediness and randomness in the search for optimal solutions [111]. The GRASP algorithm is chosen for this research due to its: (1) efficient balance of exploration and exploitation in search spaces [109], (2) ability to produce high-quality solutions efficiently, (3) easier implementation [50], and (4) proven efficiency in various optimization problems, including single machine [53], the Chinese postman routing [54], flow shop [55], and JSS [56].

As compared to other approximate methods, GRASP offers several advantages: (1) it requires only two parameters to tune (the candidate list and the number of iterations) [51], (2) it is highly scalable for JSS problems, as its computational effort does not grow exponentially with the problem size, (3) it adapts quickly to dynamic events by regenerating solutions and applying local search to accommodate the new problem state, and (4) its randomized component introduces diversification, enabling exploration of diverse solution spaces, potentially finding better solutions.

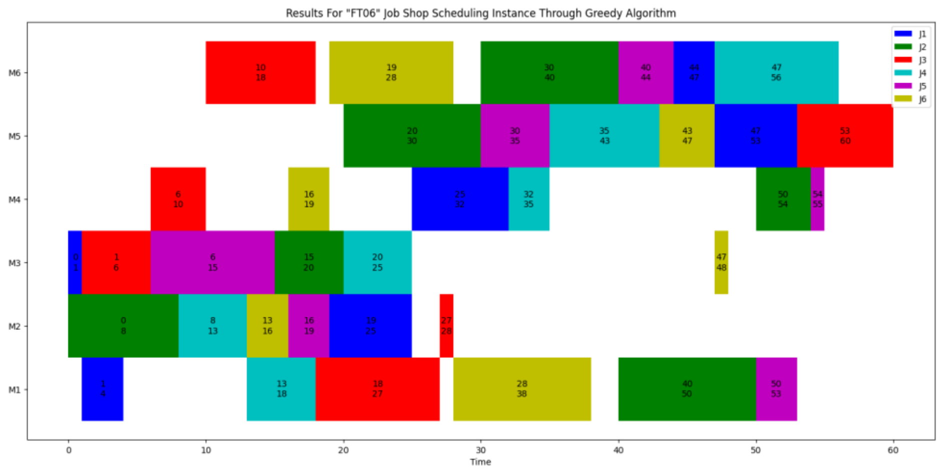

Fig. 6. Gantt chart for hypothetical ft06 bike manufacturing using G_rA

TABLE VI. RESULTS OBTAINED THROUGH GREEDY ALGORITHM FOR FT06 HYPOTHETICAL MANUFACTURING

(S_{ij}, C_{ij})	**Frame cutting and shaping (O_1)**	**Frame welding (O_2)**	**Painting the frame (O_3)**	**Assembling (O_4)**	**Inspection (O_5)**	**Packaging (O_6)**	**Tardiness (units)**
Mountain bike (J_1)	(0,1)	(1,4)	(19,25)	(25,32)	(44,47)	(47,52)	0
Road bike (J_2)	(0,8)	(15,20)	(20,30)	(30,40)	(40,50)	(50,54)	23
Hybrid bike (J_3)	(1,6)	(6,10)	(10,18)	(18,27)	(27,28)	(53,60)	4
Cruiser bike (J_4)	(8,13)	(13,18)	(20,25)	(32,35)	(35,43)	(47,56)	0
Electric bike (J_5)	(6,15)	(16,19)	(30,35)	(40,43)	(50,53)	(54,55)	13
Folding bike (J_6)	(13,16)	(16,19)	(19,28)	(28,38)	(43,47)	(47,48)	0
Computational time							0.01(sec)

GRASP consists of two phases: the construction phase, which provides an initial feasible solution that is built using a heuristic algorithm, and the local search phase, which is applied on initial solution from construction phase.

1) *Construction phase:* For our approach, G_rA has been applied to hypothetical ft06 bike manufacturing scenario to generate the initial feasible solution. G_rA constructs this solution by starting with an empty solution and keep adding operations to the partial solution until a complete feasible sequence of operations is achieved. The initial solution obtained from the construction phase is given in Table VI showing the sequence in which each machine performed operations.

2) *Local search phase:* A novel local search phase is employed on the initial solution from construction phase, using intensification and diversification concepts. Through these concepts, the algorithm explores promising regions with a more directed approach to guide the search procedure in promising regions of the solution space, rather than just random swaps as in typical GRASP. This approach focuses on the operations contributing significantly to the total tardiness values in the initial feasible solution. By directing the local search on operations with a significant tardiness contribution value, the algorithm effectively leverages for improvement within the area of the current solution. Furthermore, a random restart and random swapping procedure have been introduced to prevent the solution from getting stuck in local optima. A pseudocode is presented for the local search in Fig. 7, with the symbols used in pseudocode in Table VII.

TABLE VII. SYMBOLS USED IN PSEUDOCODE FOR GRASP

Notion	Meaning	Notion	Meaning
s	Initial solution	pre_{jk}	Predecessor
T_j	Job Tardiness	suc_{jk}	Successor
a	Max. iterations	I	Iterations
c	Restart interval	R	Random start
k	Operations	S_{best}	Current best
b	Max. stuck iterations	r_1, r_2	Random Operations
T_{jk}	Operation tardiness contribution		

The algorithm takes an initial solution from construction phase as input (line 1). Then, it computes the tardiness of each job (line 2) and the tardiness contribution of each operation (line 3). A job's tardiness is the difference between its completion time past due date and the tardiness contribution of an operation is the amount of how much an operation contributes to total tardiness. The operations are listed then in the descending order of their tardiness contribution values (line 4). The parameters of algorithms include the maximum number of iterations (line 5), the maximum number of stuck iterations (line 6), and the number of iterations after which a random restart would occur (line 7).

The random restart is a technique to avoid local optima by introducing randomization into the search process. The initial solution is set to the initial best solution (line 8). The algorithm's main loop runs for the a number of iterations (line 9). For each operation with a tardiness contribution greater than a predefined value (line 10), the algorithm searches neighborhood movements: swapping the operation with its predecessor (lines 13-19), its successor (lines 20–26) and at different positions within the same machine (lines 27–34).

```
1  Input: s //initial solution
2  Calculate: $T_j(s)$ //calculate tardiness of each job
3  Calculate: $T_{jk}(s)$ //calculate tardiness contribution
4  $T_{jk} = sort\left[T_{jk_2}, .., T_{jk_{l'}}\right]$ //list down operations
5  max_iterations = $a$
6  max_stuck = b
7  Restart_interval = c
8  $S_{best}$ = s //initial best solution
9  while iteration < $a$
10     for $k$ in $T_{jk}$: //Iterate over operations sorted
11         if $T_{jk_q} > value$
12             $find$: $pre_{jk}$,$suc_{jk}$($T_{jk_q}$) //find predecessor
13             $S_{improved} = swap(s, T_{jk_q}, pre_{jk})$ //swap
14             if ($S_{improved}$):
15                 $s = S_{improved}$ //accept current solution
16                 I={}
17             else:
18                 I+=1
19             iteration += 1 // Increment the iteration
20             $S_{improved} = swap(s, T_{jk_q}, suc_{jk})$ //swap
21             if ($S_{improved}$):
22                 $s = S_{improved}$ //accept current solution
23                 I={}
24             else:
25                 I+=1
26             iteration += 1 // Increment the iteration
27             $for$ $oth_{jk}$ in $T_{jk_q}$: //Insert at all positions
28                 $S_{improved} = swap(s, T_{jk_q}, oth_{jk})$ //swap opt.
29                 if ($S_{improved}$):
30                     $s = S_{improved}$ //accept solution
31                     I={}
32                 else:
33                     I+=1
34                 iteration += 1
35         if iteration= c
36             Select $r_1(s)$, $r_2(s)$ //randomly select operation
37             $s = swap(s, r_1, r_2)$ //swap random operation
38             I={}
39         else:
40             I +=1
41         iteration += 1
42         if $I \geq b$ //stuck reached the threshold
43             $s = R(s)$ //perform random swapping
44             I={}
45     if $S_{impoved}$:
46         $S_{improved} = s$ //accept best solution
47     iteration +=1
48 return $S_{best}$
```

Fig. 7. Pseudocode for GRASP algorithm

If any of these neighborhood movements result in a better solution (in terms of total tardiness), the current solution is updated/modified (lines 15, 21, and 30). After a certain number of iterations, the algorithm determines if a random restart (c) would occur (lines 35-41). If the solution keeps stuck (b) for a specific number of iterations with the same objective function value, random swapping will be done (lines 42-44). After exploring all operations, the algorithm updates the best solution if it is better than the previously found best solution (line 48).

3) Intensification: The algorithm first calculates each operation's contribution to total tardiness in the initial solution. It then creates a sorted list of operations in decreasing order of tardiness contribution. Iterating through this list, the algorithm prioritizes operations with high tardiness impact. It performs a series of swaps, moving these operations to positions of predecessors, successors, and other operations on the same machine. This process continues until a predefined criterion is met. After each swap, the algorithm evaluates its impact on total tardiness. If the swap minimizes total tardiness, the new solution is accepted.

4) Diversification: The algorithm employs diversification to search multiple regions of the solution space and avoiding local optima. Our algorithm has two diversification techniques: random swap, if the solution gets stuck for a certain number of iterations and random start, after a specified time. The random swap enhances diversity by incorporating randomization into operations swapping, preventing the algorithm from being stuck in local optima. Random restart allows the algorithm to explore a wider solution space, resulting in better solution.

The GRASP algorithm has been applied to a hypothetical ft06 bike manufacturing scenario. During the construction phase, an initial solution, representing the sequence of operations on each machine, is generate, refer to Table VI. GRASP then calculates the tardiness of each job in this initial solution, identifies operations with significant tardiness values, and computes their contributions to the total tardiness, as seen in Table VIII.

TABLE VIII. COMPUTING JOB TARDINESS AND TARDINESS CONTRIBUTION OF OPERATIONS

Jobs	T_j	T_{jk}
J_1	0	O_1: 0, O_2: 0, O_3: 0 , O_4: 0 , O_5: 0 , O_6: 0
J_2	21	O_1: 0, O_2: 0, O_3: 0, O_4: 7, O_5: 10, O_6: 4
J_3	0	O_1: 0, O_2: 0, O_3: 0 , O_4: 0 , O_5: 0 , O_6: 0
J_4	0	O_1: 0, O_2: 0, O_3: 0 , O_4: 0 , O_5: 0 , O_6: 0
J_5	11	O_1: 0, O_2: 0, O_3: 0 , O_4: 3 , O_5: 7 , O_6: 1
J_6	0	O_1: 0, O_2: 0, O_3: 0 , O_4: 0 , O_5: 0 , O_6: 0

From Table VIII, the road bike's (J_2) operation (O_5) on the frame cutting and shaping machine (M_1) has the highest tardiness contribution, taking 10 times unit. GRASP identifies its predecessor (folding bike's (J_6) operation O_4) and successor (electric bike's (J_5) operation O_5). It then swaps J_2's operation O_5 with its predecessor, successor, and other neighboring operations on the same machine. If an improvement is found, the solution is updated. This process repeats operations with a specific tardiness contributions value until predefined criteria are met. The Table IX presents the results obtained for hypothetical ft06 bike manufacturing company scenario in terms of S_{ij}, F_{ij}, T_j and computational time. Additionally, the final solution is visualized as Gantt chart in Fig. 8. Each bar on Gantt chart represents an operation, with the values indicating its S_{ij} and C_{ij} on that machine.

The results obtained for ft06 instance using the exact method, G_rA, GRASP and the previous studies, such as [68] and [114], are compared (as shown in Table X). While these studies used makespan as the objective function (with an optimal value of 55, consistent with our findings upon evaluation), our study focused on the total tardiness. Thus, only the comparison between computational times has been made. The Table X shows that our exact method achieved a total tardiness value of 32 in 2.11 sec. The G_rA was faster with computational time of 0.001 sec but had a higher tardiness value of 40. In contrast, GRASP achieved the optimal total tardiness value of 32 in 0.003 sec, balancing computational efficiency and solution quality. Also, previous studies on ft06 instance reported longer computational times, such as 0.37 sec in [68] and less than 1 sec in [114], as compared to our approach. These results highlight the efficiency of our novel GRASP, in solving the ft06 instance.

V. Experimental Setup

To expand the validation of the effectiveness of our exact, G_rA and GRASP, experiments were conducted considering two scenarios: (1) hypothetical JSS instances, and (2) benchmark instances from [48], [107], [115], [116]. The instances are categorized as small (≤ 25 operations, e.g., 5×5), medium (26-100 operations, e.g., 10×10), and large(>100 operations, e.g., 11×11 or greater) [68]. The evaluation has been made in terms of computational effectiveness and solution quality.

I. In terms of computational effectivenss

1) Hypothetical JSS Instances and exact method

Small to large hypothetical JSS instances from 2×2 to 20×20 were generated, where the number of operations for every job matches the total number of machines, based on the technique by [117]. Parameters (p_{ij}, r_j, and d_j) were randomly generated in MS Excel and routing for each job has been defined. The exact method was implemented in a programming software for instances up to 10×10, running each instance 10 times to account for variations. From these runs, the mean ($\bar{x}$), the standard deviation (SD) and the coefficient of variability (CV) have been calculated to evaluate the solution quality. Results were obtained only upto 9×9 instance but for 10×10 instance, despite running the optimization process in programming software for 172800 sec (48 hours), we had to stop the optimization process without reaching optimality, highlighting computational challenges for large instances in the exact method. A line chart in Fig. 9 presents visualization for computational time trends for the results obtained for small to medium instances, with each line representing a distinct test run and the bold black line showing the average. The line chart shows the exponential growth in computational time with problem size.

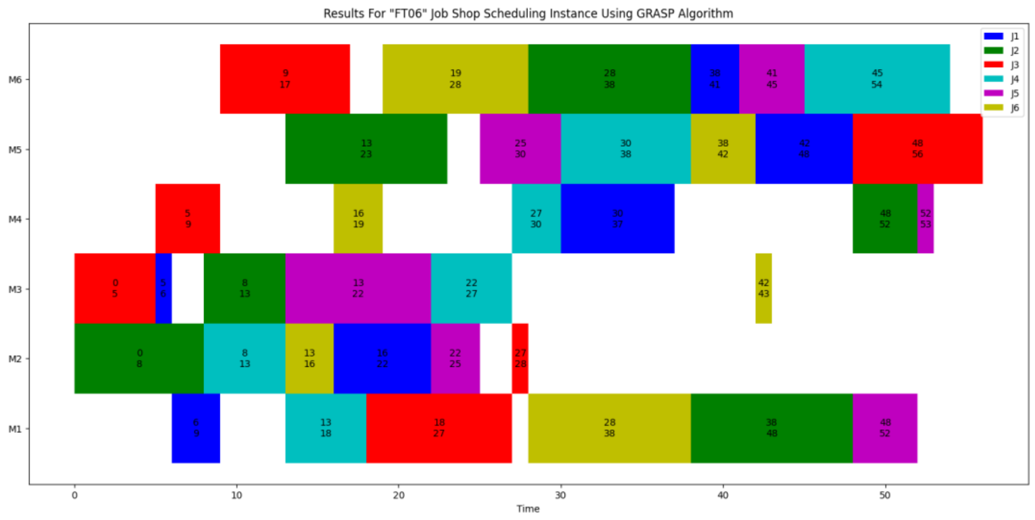

Fig. 8. Gantt chart for ft06 manufacturing using GRASP algorithm

TABLE IX. Results Obtained Through GRASP Algorithm for FT06 Hypothetical Manufacturing

(S_{ij}, C_{ij})	Frame cutting and shaping (O_1)	Frame welding (O_2)	Painting the frame (O_3)	Assembling (O_4)	Inspection (O_5)	Packaging (O_6)	Tardiness (units)
Moutain bike (J_1)	(5,6)	(6,9)	(16,22)	(30,37)	(38,41)	(42,48)	0
Road bike (J_2)	(0,8)	(8,13)	(13,23)	(28,38)	(38,48)	(48,52)	21
Hybrid bike (J_3)	(0,5)	(5,9)	(9,17)	(18,27)	(27,28)	(48,56)	0
Cruiser bike (J_4)	(8,13)	(13,18)	(22,27)	(27,30)	(30,38)	(45,54)	0
Electric bike (J_5)	(13,22)	(22,25)	(25,30)	(41,45)	(48,52)	(52,53)	11
Folding bike (J_6)	(13,16)	(16,19)	(19,28)	(28,38)	(38,42)	(42,43)	0
Computational time							0.003 (sec)

TABLE X. COMPARISON OF RESULTS FOR EXACT METHOD, G_rA AND GRASP WITH OTHER STUDIES FOR FT06 JSS INSTANCE

Methods	Computational time	Objective function
Exact method	2.11	32
G_rA algorithm	0.001	40
GRASP algorithm	0.003	32
GRASP (in [68])	0.37	-
GRASP (in [114])	<1	-

2) *Benchmark instances and exact method*

Benchmark instances such as ft10, abz5, abz6, orb01, orb02, orb03, orb04, orb05, ta01, ta02, and ta03 are medium to large instances. The exact methods prove impractical for these instances, just like 10×10 instance from hypothetical JSS instance, due to computational time constraints. This limitation necessitates exploring alternative techniques like G_rA to obtain near optimal solutions within reasonable time for medium to large instances.

3) *Hypothetical JSS instances and G_rA*

Using the data generated for small to large hypothetical instances, $\mathrm{G_rA}$ was tested using programming software, with each instance run 10 times. Based on the results obtained from G_rA, a line chart has been drawn, as shown in Fig. 10, to compare the computational effectiveness between the exact method and G_rA for small to medium instances. The line chart shows an exponential growth in computational time for small to medium instances when using the exact method. However, this significantly reduced when using G_rA. It confirms the computational effectiveness of G_rA in finding faster solutions than the exact method for small to medium instance. For the medium to large instances, G_rA yielded consistent total tardiness value with varying but significantly smaller computational time each run (refer to Table XIII).

4) *Benchmark instances and G_rA*

The same experimental procedure was applied to benchmark instances using G_rA. The results are presented in Table XIV, showing consistent total tardiness and varying but significantly smaller computational times than exact method.

5) *Hypothetical JSS instances and GRASP*

The GRASP algorithm was tested, with each instance run 10 times. Across each run, the obtained results for GRASP for the total tardiness and computational time showed variability but significantly smaller than the exact method and G_rA. It was expected due to GRASP's randomized construction and local search phase. A comparison between computational times for small to medium instances obtained from the exact method, G_rA and GRASP has been presented in Fig. 11. It can be seen from Fig. 11 that GRASP shows significant reduced computational times compared to the exact method, though slightly higher but still acceptable compared to G_rA. For example, for 9×9, the computational time for the exact method is 23758.62 sec, G_rA is $1.34e^{-3}$ sec and GRASP is $2.18e^{-1}$ sec. Similarly, for medium to large instances, results are presented in Table XIII, showing that GRASP consistently outperforms G_rA in solution quality and the exact method in terms of computational effectiveness.

6) *Benchmark instances and GRASP*

We also conducted experiments for GRASP using benchmark instances. The results obtained from these experiments are presented in Table XIV, showing a similar trend of significantly minimizing the computational time than the exact method, as with GRASP in hypothetical instances.

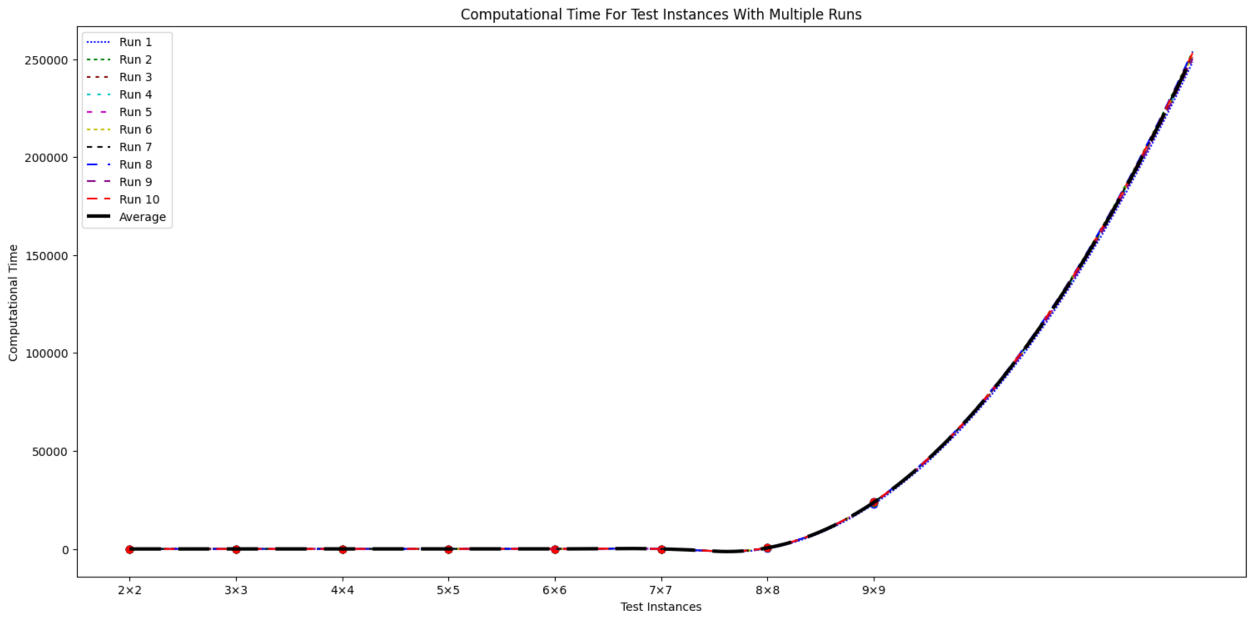

Fig. 9. Computational times obtained using exact method for a range of hypothetical test instances for JSS

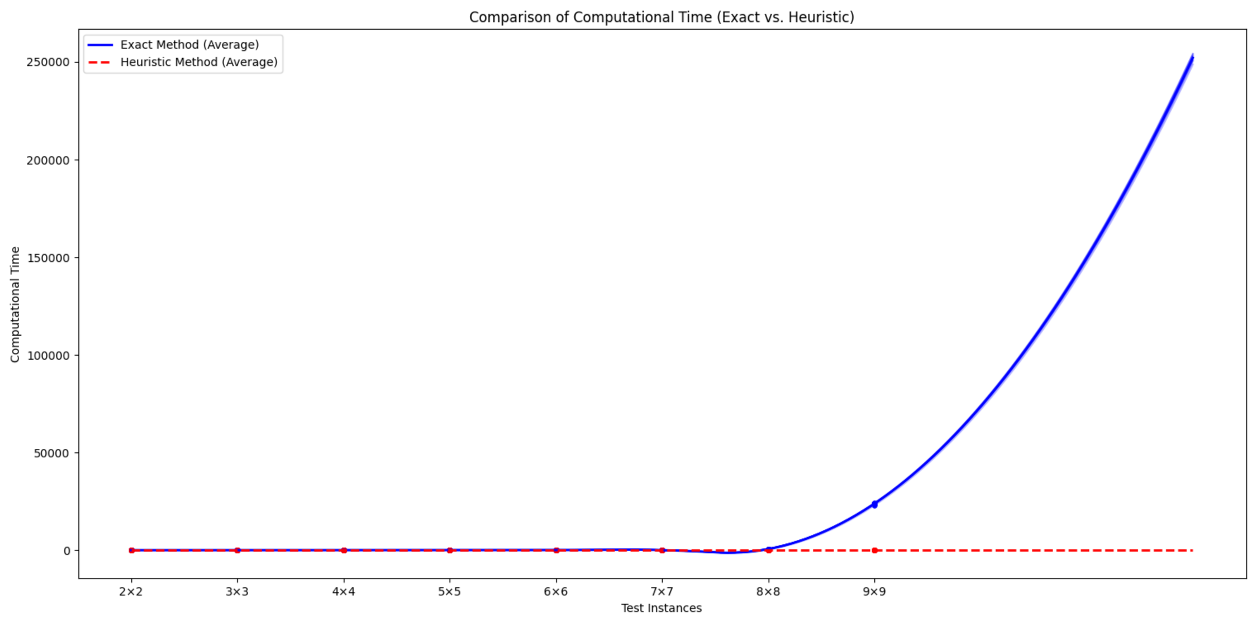

Fig. 10. Computational time comparison between exact method and G_rA for hypothetical test instances

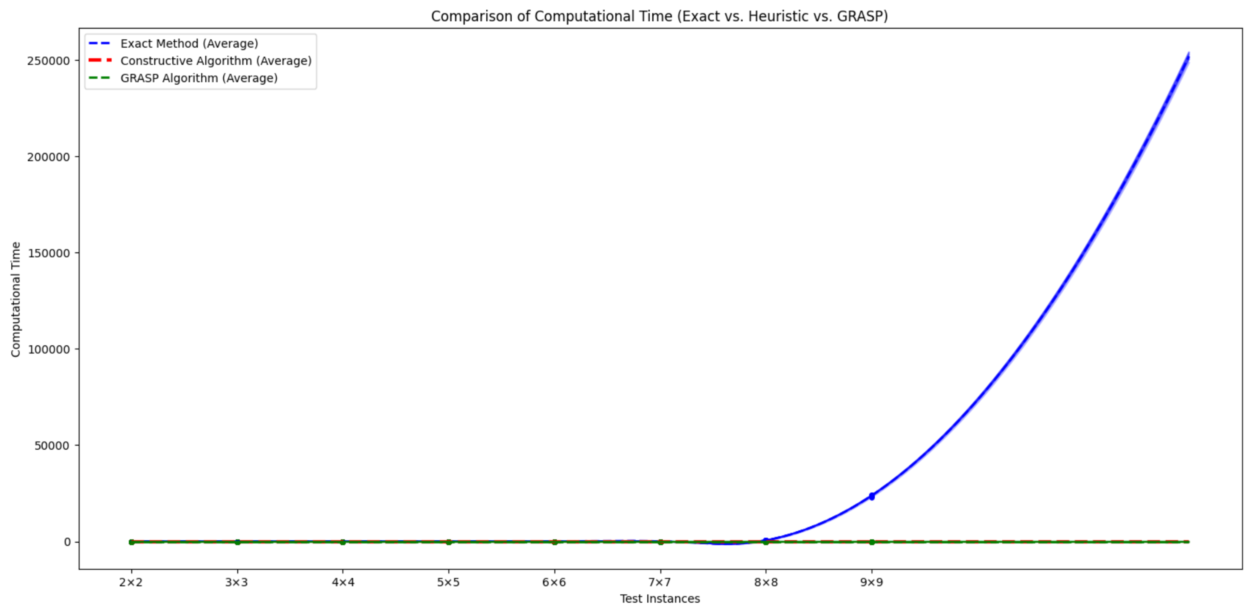

Fig. 11. Computational time comparison between exact method, GA and GRASP algorithm for hypothetical test instances

J. *In Terms of Solution Quality*

In addition to evaluating computational efficiency, it is crucial to evaluate the solution quality in JSS. High-quality solutions ensure that production goals are achieved. To achieve this, we calculated the percentage gap between the exact method and the G_rA, as well as between the exact method and the GRASP. This comparison is critical to determine how closely each solution approximates the optimal solutions provided by the exact method.

1) *Exact method and G_rA*

Fig. 12 presents a bar chart showing the percentage gap between the exact method and G_rA (blue), and the exact method and GRASP (red) for small to medium instances. The bar chart reveals that while the G_rA is computationally efficient, it has an average percentage gap of 13.82%. This means that, on average, the solutions generated by the G_rA are 13.82% less optimal compared to the exact method.

2) *G_rA and GRASP*

In contrast to percentage gap between the exact method and G_rA, the GRASP shows a remarkable improvement with an average percentage gap of only 3.43%, highlighting its superior accuracy in producing solutions closer to the optimal (refer to Fig. 12).

Table XIII presents percentage gap across various test instances, consistently showing that the GRASP algorithm outperforms the G_rA in terms of solution quality. For instance, in the 4×4 the G_rA has a percentage gap of 20.12%, whereas the GRASP reduced this to 0.00%. Similarly, in the 8×8 instance, the percentage gap dropped from 19.85% for G_rA to 5.83% for GRASP. These results highlight the reliability of the GRASP in maintaining high solution quality across different problem sizes.

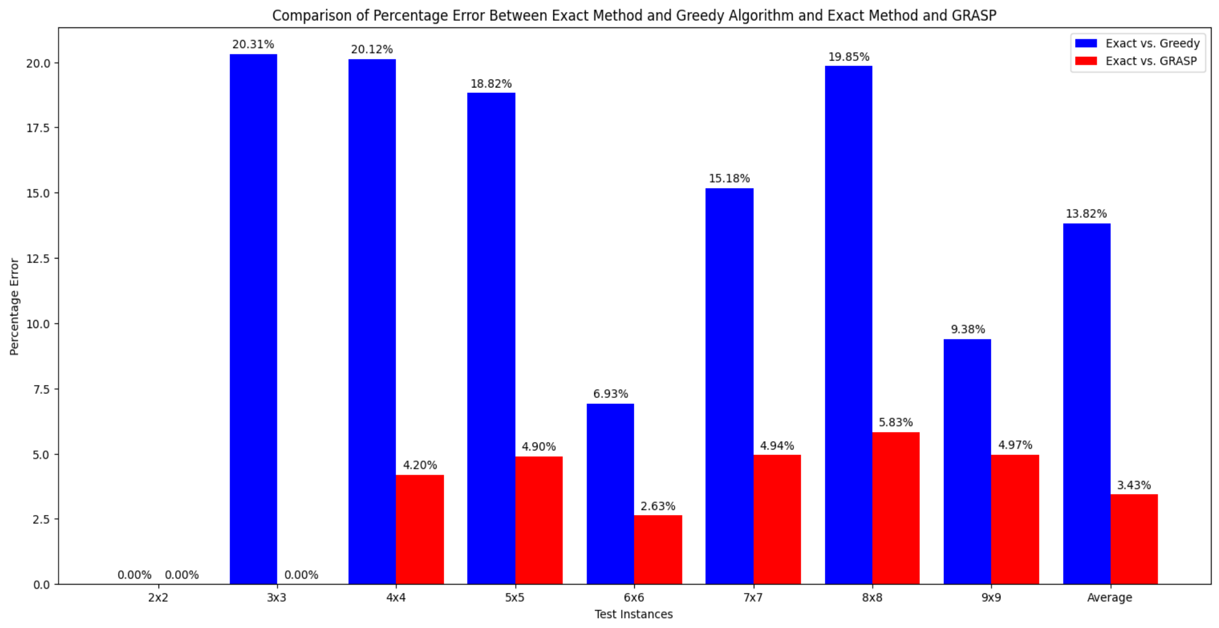

Fig. 12. Percentage error for G_rA and GRASP with exact method

VI. Rescheduling Procedure

Since production environments are often dynamic and unpredictable, where they experience dynamic events [32], [33], [34], that deviate the original schedule, reducing the efficiency and quality of scheduled execution [28]. To tackle them, rescheduling has emerged as a primary focus of contemporary scheduling research and a topic of global interest [36].

In order to bring together the various aspects of rescheduling, which have often been studied separately in previous research, a comprehensive rescheduling framework has been presented, as seen in Fig. 13. Our framework combines multiple rescheduling elements, including rescheduling factors, strategies, policies, methods, environments, and performance evaluation. This holistic approach provides a more complete understanding of the rescheduling process, which is crucial for both theoretical development and practical applications.

In our study, we examined a dynamic scenario at hypothetical ft06 bike manufacturing company. The production environment of the case example is subjected to SDE (i.e., NJA, RO, MF and SMM) necessitating rescheduling strategy. To address this, we have implemented a $P_{act} - R_{act}$ strategy that incorporates RF and Reg methods using a Hyb (p_e and ED) policy. The adopted rescheduling methodology is presented as a flowchart in Fig. 15. The rescheduling methodology starts with obtaining the initial schedule, which is then executed in production environment. The algorithm continuously monitors for any SDE. When such an event occurs, it is categorized as either machine-related, job-related, or other.

For machine-related events, the algorithm further categorizes if the event is a MF. If yes, an ED policy is immediately implemented using a RF method. This involves forwarding the schedule to account for the disruption, computing the new machine availability status and time, and determining the remaining processing time for jobs. If the machine-related event is SMM, a p_e policy is applied at predefined rescheduling points. For job-related events, the algorithm checks if the event is a RO or NJA. If it is a RO, an ED policy is applied, which schedules RO immediately. If it is NJA, a p_e policy is implemented at the rescheduling point.

For rescheduling jobs upon dynamic events, the algorithm computes the machine and job availability statuses and times at the rescheduling point. Completed operations are removed from the new schedule, and a new rescheduling scheme is created for the remaining jobs while including newly arriving jobs. The algorithm then computes new start times for each operation based on the job and machine availability times and executes the new scheduling plan. This process continues iteratively, with the algorithm re-evaluating and rescheduling as needed until all operations are finished, at which point the process ends.

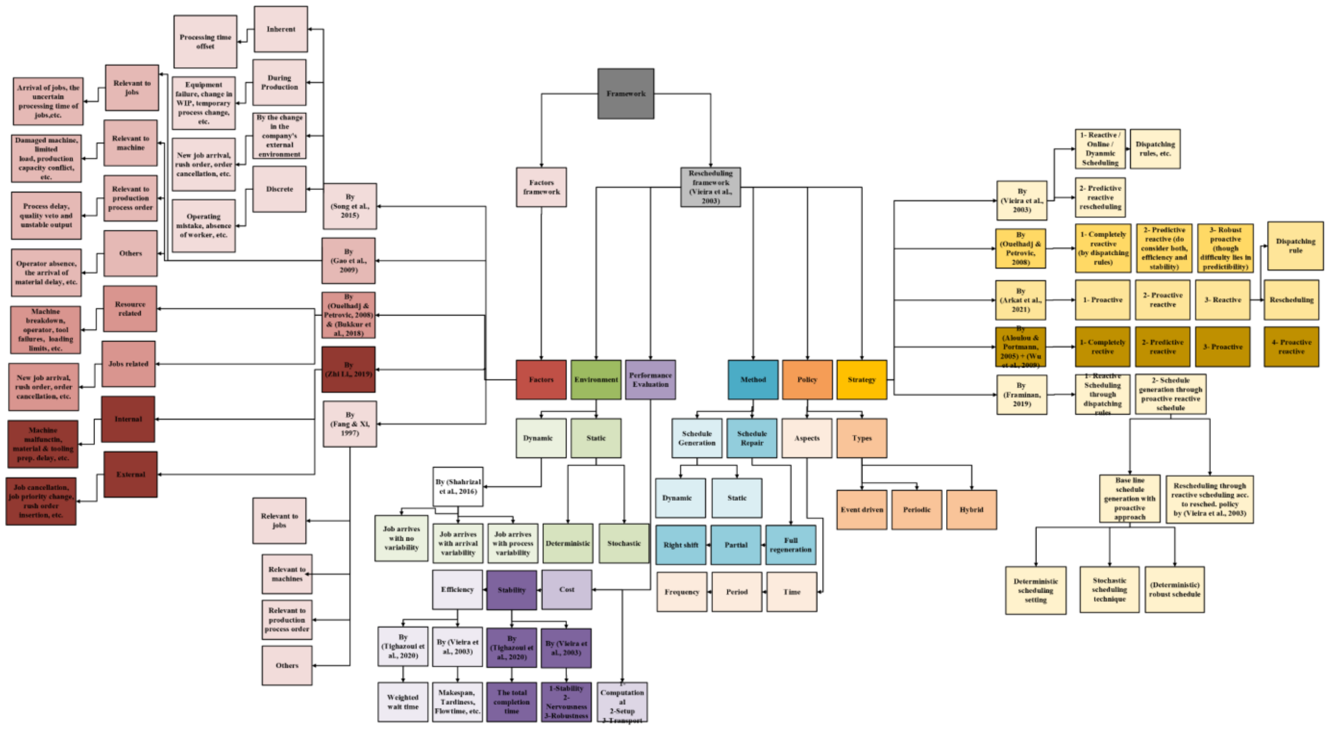

Fig. 13. Rescheduling framework [36], [37], [38], [43], [46], [85], [87], [95], [97], [102], [105], [106], [119]

K. *Periodic rescheduling* (p_e)

In this study, p_e policy is implemented by dividing the entire time horizon into specific intervals, known as rescheduling points. A pseudocode for p_e has been presented in Fig. 14, where the decision criteria involve determining the next rescheduling point R_q. Jobs arriving after R_{q-1} are ignored until the next rescheduling point R_q is reached. At R_q, the algorithm reschedules the jobs. The schedule is then updated to reflect these changes. The algorithm terminates when all jobs finished process.

```
1   Initialize $q = 0$
2   Initialize $R_q = 0$
3   while NotAllJobsProcessed:
4       $q = q + 1$
5       $R_q$=determineReschedulingPoint ($R_{q-1}$)
6       IgnoreJobsArrivingAfter ($R_{q-1}$)
7       ReschedulingJobsAt $R_q$
8       If newJobsArrivingAfter $R_{q-1}$
9           RescheduleJobsAt($R_q$)
10      UpdateSchedule()
11  end
```

Fig. 14. Pseudocode for periodic rescheduling

1) *New job arrival*

Three rescheduling points are considered: $t = 15$, $t = 30$, and $t = 45$ to schedule the NJA.

a) Case I: New mountain bike arrival: At time $t = 12$, a new job involving the production of mountain bikes arrives. According to the p_e policy, this job is delayed until the next rescheduling point at $t = 15$. At $t = 15$, a Reg method is employed. The algorithm first calculates the availability of machines and jobs on their status at $t = 15$ (shown in Table XVI). Based on this information, it determines the earliest start times for each job. Operations that have already been completed by $t = 15$ are excluded from the rescheduling process. The jobs that were still processing when rescheduling point arrived, are neglected. The newly arrived jobs, along with the unfinished operations of existing jobs, are included in the rescheduling procedure in Fig. 15.

During the periodic rescheduling process at $t = 15$, a situation arose where the algorithm needed to select between two jobs (J_4 and J_7), which are scheduled to start on the same machine (M_3) at the same time ($t = 22$). To address this, we examined five priority rules: Earliest Due Date (EDD), Remaining Operations, Remaining Processing Time, First in First Out (FIFO), and Critical Ratio (CR). Although these rules produced similar results with total tardiness value of 35 for each scenario, our approach focused on EDD due to its proven effectiveness in minimizing job tardiness.

b) Case II: Regular interval with no new jobs: At the second rescheduling point, $t = 30$, no new jobs arrive. As a result, the scheduling process continues uninterrupted, and the system maintains its current schedule.

c) Case III: Subsequent new job arrival: At time $t = 40$, another new job arrives, this time involving the production of electric bikes. Like the previous case, this job is delayed until the next rescheduling point at $t = 45$. At this point, the algorithm again calculates machine and job availability to determine the earliest start times for all jobs, like the rescheduling procedure for new job arrival at $t = 15$. Completed operations are removed, while new and remaining jobs are included in the Reg rescheduling process. The algorithm also checks if two or more jobs start their operations on the same machine at the same time after the rescheduling process.

The final solution incorporating new jobs arrival at $t = 15$ and $t = 45$ can be visualized in Gantt chart that has been provided in Fig. 16. The results demonstrate that the p_e policy effectively accommodates new jobs by rearranging the schedule dynamically at rescheduling points.

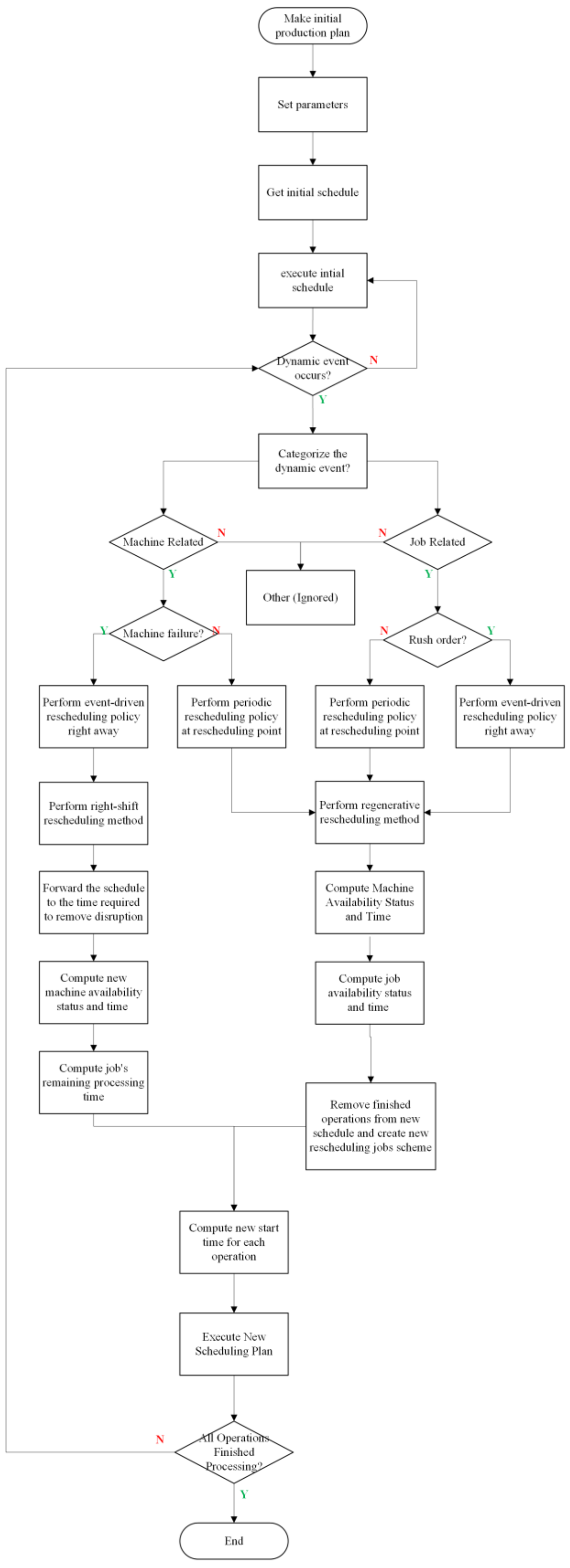
Make initial production plan
Set parameters
Get initial schedule
execute intial schedule
Dynamic event occurs?
N
Y
Categorize the dynamic event?
Machine Related
N
N
Job Related
Other (Ignored)
Y
Y
Machine failure?
Y
N
N
Rush order?
Y
Perform event-driven rescheduling policy right away
Perform periodic rescheduling policy at rescheduling point
Perform periodic rescheduling policy at rescheduling point
Perform event-driven rescheduling policy right away
Perform right-shift rescheduling method
Perform regenerative rescheduling method
Forward the schedule to the time required to remove disruption
Compute Machine Availability Status and Time
Compute new machine availability status and time
Compute job availability status and time
Compute job's remaining processing time
Remove finished operations from new schedule and create new rescheduling jobs scheme
Compute new start time for each operation
Execute New Scheduling Plan
All Operations Finished Processing?
N
Y
End

Fig. 15. Proposed rescheduling procedure

2) Scheduled machine maintenance (SMM)

For the SMM, a p_e policy is again employed. In this scenario, M_4 is scheduled for maintenance at time $t = 30$, requiring 10 times unit to complete the maintenance task. This means that no job can be scheduled on M_4 during the maintenance period from $t = 30$ to $t = 40$. At $t = 30$, the M_4 is goes under maintenance and jobs that were supposed to be processed on M_4 during that time are delayed accordingly. For example, operation O_4 of J_1 was scheduled on M_4 at $t = 30$ but it will be delayed until $t = 40$. The results of SMM are visualized using Gantt chart, as shown in Fig. 17.

L. *Event-driven* (ED) *Rescheduling*

Two SSE are considered for event-driven rescheduling: MF, and RO. These events necessitate immediate adjustments to the production schedule.

1) Machine failure (MF)

For MF, the RF method is applied. This method involves shifting the operations scheduled on the failed machine forward in time to account for the downtime.

a) Scenario 1: Machine 3 (M_3) *fails:* In this scenario, at time $t = 20$, machine M_3 fails while job J_5 is processing on it. Job J_5 started its operation on M_3 at $t = 13$ and required 9 times unit to complete, but the machine broke down at $t = 20$. Machine M_3 needs 10 time unit to recover and will be available again at $t = 30$. The RF method is employed on machine M_5 which right shifts the operation of J_5 on M_5 and the remaining processing time for job J_5 is recalculated.

Since J_5 had processed for 7-time unit before the breakdown (from $t = 30$ to $t = 20$.), it needs 2 more-time units to complete its operation. Job J_5 restarts its operation on M_3 at $t = 30$, immediately after the machine is repaired, and completes at $t = 32$.

b) Scenario 2: Machine 6 (M_6) *fails:* At time $t = 46$, machine M_6 fails while job J_1 is processing on it. Job J_1 started its operation on M_6 at $t = 45$ and required 3-time units to complete, but the machine broke down at $t = 46$. The RF method is again employed, which right-shifts the operations of J_1 by the amount of time required to repair the machine. Machine M_6 needs 10 time units to restore the machine and will be available again at $t = 56$. The remaining processing time for job J_1 is recalculated.

Since J_1 had processed for 1 time unit before the breakdown (from $t = 45$ to $t = 46$), it needs 2 more-time units to complete its operation. Job J_1 restarts its operation on M_6 at $t = 56$, immediately after the machine is repaired, and completes at $t = 58$. The results of machine failure are visualized using Gantt chart, as shown in Fig. 18.

2) Rush orders (RO)

RO are urgent jobs that need to be incorporated into the current schedule with high priority. The rescheduling process for RO involves evaluating the current machine availability and job availability and integrating the RO into the production flow in order to minimize total tardiness.

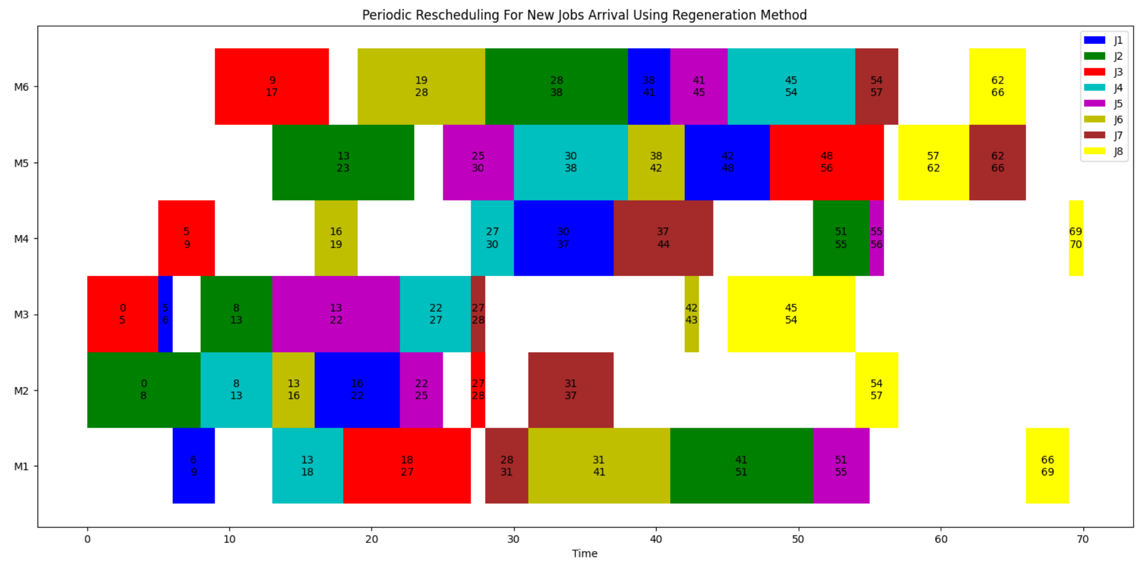

Fig. 16. Periodic rescheduling for new jobs arrival at $t = 15, 30$ and 45

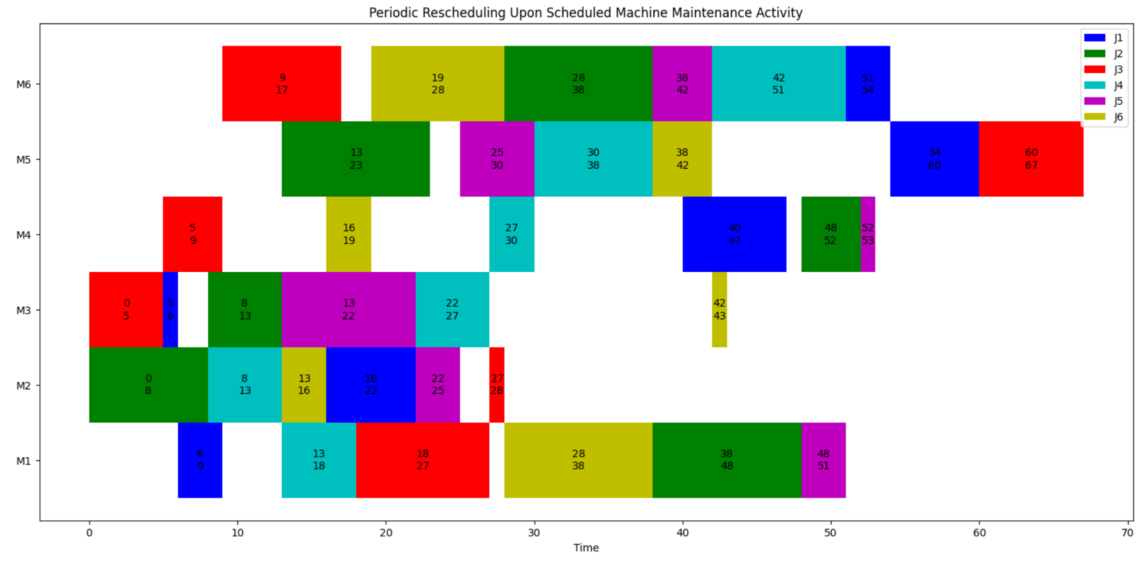

Fig. 17. Periodic rescheduling at t=30 for machine scheduled maintenance

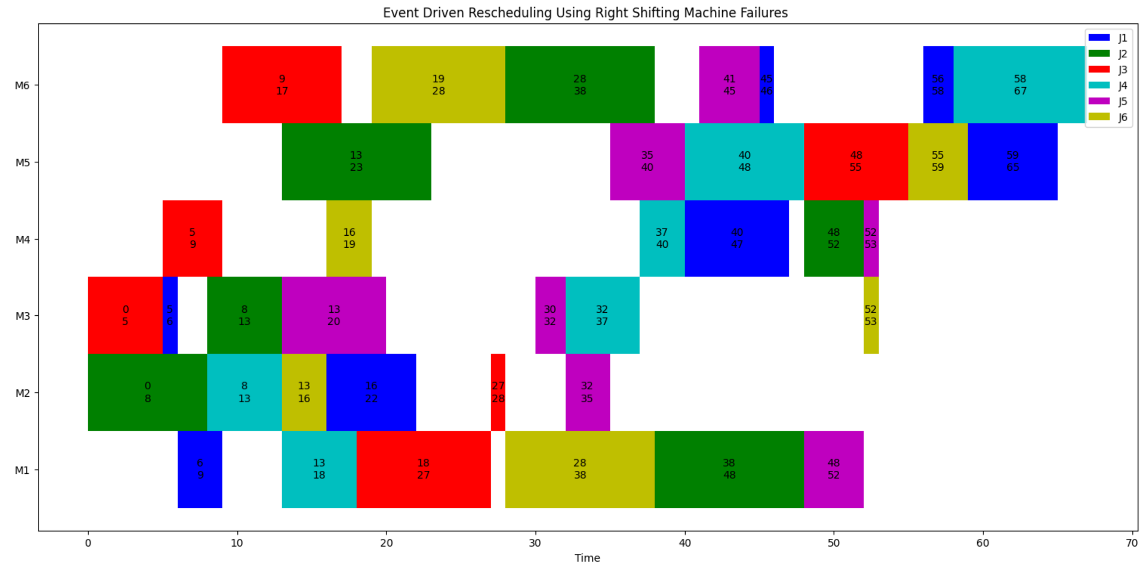

Fig. 18. Event-driven rescheduling at t=20 and t=46 for M3 and M6 failure using right-shift rescheduling methods

a) Scenario 1: Hybrid bike rush order: A RO for a hybrid bike arrives at $t = 35$. ED policy is required based on the routing for the hybrid bike. At $t = 35$, the algorithm first calculates the availability of machines and jobs. The earliest start times for each operation of the hybrid bike are computed, considering the machine's availability and the job's routing requirements, without waiting for the next rescheduling point to arrive. The schedule is updated accordingly to reflect the inclusion of the RO.

b) Scenario 2: Electric bike rush order: the second RO of electric bike arrives at $t = 40$. The algorithm again calculates the machine and job availabilities and based on that, it calculates the earliest start time, at the point where the event arrived, without waiting for the periodic rescheduling point to arrive and the schedule is updated to integrate the new RO. The results of RO are visualized using Gantt chart, as shown in Fig. 19.

Finally, the Table XVII compares different rescheduling methods applied to a hypothetical ft06 bike manufacturing company under various dynamic events. The focus is on the total tardiness value before and after rescheduling, which serves as a key performance metric.

R_{act} strategies tend to slightly increase tardiness when responding to SSE and increasing total tardiness from 32 to 33 for RO and to 38 for MF. Proactive strategies, however, result in higher tardiness when integrating NJA or accommodating SMM. This trend aligns with the findings presented in research [68], where the ED policy provided better results than the p_e policy.

VII. Results and Discussions

This section presents the main findings for the exact method, G_rA, and GRASP in terms of: (1) solution quality, and (2) computational efficiency. The findings are divided into five parts:

(1) Results based on small to medium hypothetical JSS instances,

(2) Results based on medium to large hypothetical JSS instances,

(3) Results based on small to large well-known benchmark JSS instances,

(4) Results comparison with the similar studies in literature,

(5) Strengths and limitations.

M. Results Based on Small to Medium Hypothetical JSS Instances

For small to medium hypothetical JSS instances: Table XI compares total tardiness values and percentage gap obtained from the exact method, G_rA and GRASP. Experiments showed the exact method provided optimal solutions, serving as a benchmark for G_rA and GRASP (refer to Table XI). Both, G_rA and GRASP achieved optimal or near-optimal solutions for smaller instances (up to 4×4), GRASP outperformed G_rA for larger instances. The percentage gap between the exact method and G_rA increased with problem size, for example, 15.18% for the 7×7 instance. In contrast, GRASP maintained a relatively low percentage error, with a maximum of 4.94% for the 7×7 instance, demonstrating better solution quality.

Computationally, the exact method was efficient for small instances but grew exponentially, taking about 23758.62 sec (around 6.6 hours) for the 9×9 instance (refer to Table XII). The results for 10×10 could not reach optimality even after running the optimization process for 48 hours.

The G_rA, on the other hand, showed significantly lower computational times, for small to medium instances being in the order of e^{-4} sec (as seen in Table XII). For small to medium instance, G_rA's computational time was only $1.34e^{-3}$ sec. Both G_rA and the exact method displayed low SD and CV, indicating consistent computational times across runs.

GRASP balanced computational efficiency and solution quality between the exact method and G_rA (refer to Table XII). For smaller instances, GRASP's times were comparable to the exact method, around e^{-2} sec or less. As problem size increased, GRASP maintained low computational times, with a maximum of 0.218 seconds for the 9×9 instance, staying within a reasonable range. GRASP showed slightly higher standard deviation in computational times due to its randomized nature. Despite this variability, GRASP outperformed G_rA in terms of solution quality. With a lower average percentage error (3.43%) compared to G_rA (13.82%), GRASP demonstrated superiority in maintaining high quality solutions while managing computational efficiency.

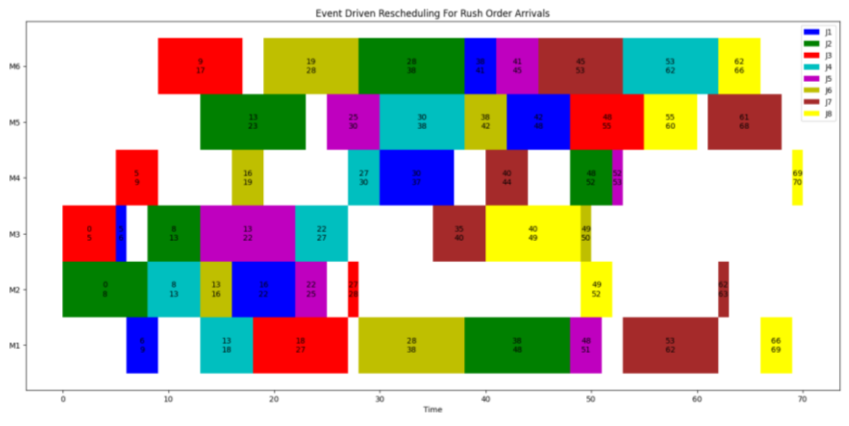

Fig. 19. Event-driven rescheduling for hybrid bike and electric bike arrival at t=30 and t=40

TABLE XI. COMPARISON OF TOTAL TARDINESS VALUES BETWEEN EXACT METHOD, GREEDY ALGORITHM, AND GRASP FOR SMALL TO MEDIUM HYPOTHETICAL JSS INSTANCES

Test Instance	EM	GA	GRASP	EM	GA	GRASP	EM	GA	GRASP	percentage Gap	
	$\bar{x}$			SD			CV			$EM\&GA$	$EM\&GRASP$
2×2	14	14	14	0.00	0.00	0.00	0.00	0.00	0.00	0.00%	0.00%
3×3	64	77	64	0.00	0.00	0.00	0.00	0.00	0.00	20.31%	0.00%
4×4	169	203	176.1	0.00	0.00	9.70	0.00	0.00	0.06	20.12%	4.20%
5×5	271	322	284.3	0.00	0.00	10.27	0.00	0.00	0.04	18.82%	4.90%
6×6	361	386	370.5	0.00	0.00	4.65	0.00	0.00	0.01	6.93%	2.63%
7×7	593	683	622.3	0.00	0.00	7.00	0.00	0.00	0.01	15.18%	4.94%
8×8	821	984	868.9	0.00	0.00	21.92	0.00	0.00	0.02	19.85%	5.83%
9×9	1066	1166	1119	0.00	0.00	27.99	0.00	0.00	0.02	9.38%	4.97%
Avg.	-	-	-	-	-	-	0.00	0.00	0.02	13.82%	3.43%
EM: exact method; GA: greedy algorithm; $\bar{x}$: mean; SD: standard deviation; CV: coefficient of variation											

TABLE XII. COMPARISON OF COMPUTATIONAL VALUES BETWEEN EXACT METHOD, GREEDY ALGORITHM, AND GRASP FOR SMALL TO MEDIUM HYPOTHETICAL JSS INSTANCES

Test Instance	EM	GA	GRASP	EM	GA	GRASP	EM	GA	GRASP
	$\bar{x}$			SD			CV		
2×2	$8.0e^{-2}$	$1.00e^{-4}$	$6.88e^{-3}$	$1.33e^{-2}$	$3.173e^{-4}$	$2.241e^{-3}$	$1.67e^{-1}$	$3.16e^{-4}$	$3.26e^{-1}$
3×3	$8.2e^{-2}$	0.00	$4.26e^{-2}$	$9.19e^{-3}$	0.00	$5.62e^{-2}$	$1.12e^{-1}$	0.00	1.31
4×4	$1.53e^{-1}$	$1.67e^{-4}$	$3.02e^{-2}$	$4.92e^{-2}$	$2.70e^{-4}$	$2.92e^{-3}$	$3.21e^{-1}$	1.62	$9.70e^{-2}$
5×5	$2.99e^{-1}$	$4.21e^{-4}$	$4.31e^{-2}$	$3.16e^{-3}$	$4.90e^{-4}$	$6.11e^{-3}$	$1.05e^{-2}$	1.16	$1.41e^{-1}$
6×6	$7.11e^{-1}$	$4.51e^{-4}$	$5.61e^{-2}$	$3.93e^{-3}$	$4.98e^{-4}$	$2.82e^{-3}$	$5.52e^{-2}$	1.10	$5.02e^{-2}$
7×7	2.03	$9.02e^{-4}$	$8.94e^{-2}$	$1.82e^{-2}$	$4.59e^{-4}$	$6.24e^{-3}$	$8.99e^{-3}$	$5.09e^{-1}$	$6.97e^{-2}$
8×8	553.092	$9.57e^{-4}$	$1.64e^{-1}$	115.39	$1.56e^{-4}$	$2.16e^{-2}$	$2.08e^{-1}$	$1.63e^{-1}$	$1.31e^{-1}$
9×9	23758.62	$1.34e^{-3}$	$2.18e^{-1}$	322.46	$4.82e^{-4}$	$1.81e^{-2}$	$1.35e^{-2}$	$3.58e^{-1}$	$8.30e^{-2}$
Avg.	-	-	-	-	-	-	0.11	1.00	0.27

N. Results Based on Medium to Large Hypothetical JSS Instances

For medium to large hypothetical JSS instances, G_rA and GRASP gained significance due to the computational limitations of the exact methods. A comparative analysis for instances ranging from 10×10 to 20×20 showed that G_rA 's maintained low computational times (from $1.96e^{-3}$ sec for the 10×10 instance to $1.94\ e^{-2}$ sec for the 20×20 instance) (refer to Table XIII). Additionally, G_rA 's low SD and CV indicated consistent computational times across multiple runs. In contrast, GRASP exhibited computational times close to those of the G_rA, ranging from $1.48e^{-1}$ sec for the 10×10 instance to 48.76 sec for the 20×20 instance. Although GRASP showed higher variability in computational times, as indicated by slightly higher standard deviations and coefficients of variation, this is expected due to its randomized nature and sophisticated search procedures. Despite this, GRASP outperformed G_rA in solution quality.

The mean total tardiness values for G_rA ranges from 1937 for the 10×10 instance to 6697 for the 20×20 instance, with no variability across runs. While GRASP consistently achieved lower total tardiness values compared to G_rA, with mean values ranging from 1296.2 for the 10×10 instance to 6080.5 for the 20×20 instance. Although GRASP showed non-zero SD, indicating variability in solution quality across multiple runs, it consistently delivered better results than G_rA. This highlights GRASP's superiority in handling larger-scale JSS instances, balancing computational efficiency with higher solution quality.

O. Results Based on Small to Large Well-Known Benchmark JSS Instances

To further validate the efficacy and generalize the findings of the proposed scheduling approaches, we conducted an extensive evaluation using benchmark JSS instances. The instances considered include abz5, abz6, ft10, orb01, orb02, orb03, orb04, orb05, ta01, ta02, and ta03, which encompass varying sizes and complexities.

Table XIV demonstrates $G_rA's$ consistent computational times across all instances ($1.00e^{-3}$ sec for the ft10 instance to $9.74e^{-4}$ sec for the orb05 instance), indicating stable and efficient performance in terms of computational effectiveness. GRASP, despite its sophisticated search mechanism, maintains similarly small times close to those of G_rA, ranging from 2.78 sec for the ta02 instance to $6.73e^{-1}$ sec for the orb05 instance.

In terms of solution quality, the GRASP algorithm consistently outperformed the G_rA across all benchmark instances by achieving lower total tardiness values (refer to Table XIV). For example, for abz6, the mean total tardiness value was minimized from 7614 with G_rA to 5067.7 with GRASP. Similarly, for orb04, it minimized from 8444 with G_rA to 5104.1 with GRASP. This trend held across other benchmark instances as well, showing GRASP's superior performance and effectiveness in providing high quality solutions, especially for larger and more complex instances.

In summary, for small and medium-sized instances, the exact method is preferable due to its ability to find optimal solutions, despite longer computational times compared to the G_rA (refer to Table XI and Table XII). However, as the problem size increases, the computational advantage of the G_rA becomes more pronounced, albeit compromised solution quality (refer to Table XIII and Table XIV). GRASP emerged as a practical alternative for larger instances, providing near-optimal solutions within reasonable computational times, outperforming the G_rA. Thus, while G_rA is suitable for limited computational resources, GRASP is recommended for achieving high-quality solutions in larger and more complex scheduling environments within reasonable amount of time.

P. Results Comparison with the Similar Studies in Literature

We provided a comprehensive comparison between existing studies and our study demonstrating significant improvements in computational efficiency (refer to Table XV). For example, the computational time for the ft06 instance was reduced from 0.37 sec in previous studies [114] to 0.003 sec. Similarly, for the ft10 instance, our approach reduced the time from 173.2 sec for [68] to $6.19e^{-1}$ sec in our study. This trend of improved computational efficiency is consistently observed across all orb instances, with times notably lower than those previously reported.

While our study used total tardiness as the objective function instead of makespan, solution quality was confirmed through comparative analysis using hypothetical JSS instances (refer to Fig. 12 and Table XI). The results demonstrate our approach's effectiveness in achieving high-quality solutions with significantly reduced computational times, offering practical insights into GRASP method's scalability and efficiency.

Q. Strengths and Limitations

One of the strengths of this study is the thorough comparison across a wide range of instance sizes, which offers a clear understanding of each method's performance. The inclusion of benchmark instance results and comparisons with previous studies adds robustness to our findings. However, the study's reliance on hypothetical data may not fully capture the complexities of real-world scheduling problems. Challenges include integrating real-time data with the rescheduling procedure, which can be improved using IoT devices and sensors. Decision-making under uncertainty is also critical in rescheduling for which more dynamic events could be incorporated into the model. Future research could:

(1) Incorporate more dynamic events for decision-making under uncertainty,

(2) Extend analysis to real-world data with diverse objective functions (e.g., makespan, flow time), and

(3) Leverage predictive analytics (machine learning, digital twins) to forecast disruptions.

TABLE XIII. COMPARISON OF COMPUTATIONAL TIME AND OBJECTIVE FUNCTION BETWEEN GREEDY ALGORITHM, AND GRASP FOR HYPOTHETICAL LARGE INSTANCES

	GA						GRASP					
	Computational Times			Objective Function			Computational Time			Objective Function		
Instances	$\overline{x}$	SD	CV	$\overline{x}$	SD	CV	$\overline{x}$	SD	CV	$\overline{x}$	SD	CV
10×10	$1.96e^{-3}$	$5.58e^{-4}$	$2.85e^{-1}$	1937	0	0	3.54	$1.48e^{-1}$	$4.19e^{-2}$	1296.2	14.41	$1.11e^{-2}$
11×11	$2.58e^{-3}$	$4.83e^{-4}$	$1.87e^{-1}$	1653	0	0	4.93	$1.38e^{-1}$	$2.80e^{-2}$	1545.2	14.14	$9.15e^{-3}$
12×12	$3.40e^{-3}$	$4.53e^{-4}$	$1.33e^{-1}$	2048	0	0	6.87	$2.25e^{-1}$	$3.28e^{-2}$	1870.5	19.35	$1.03e^{-2}$
13×13	$4.34e^{-3}$	$4.78e^{-4}$	$1.01e^{-1}$	4783	0	0	9.52	$5.83e^{-1}$	$6.13e^{-2}$	3020.4	88.92	$2.94e^{-2}$
14×14	$7.91e^{-3}$	$5.45e^{-3}$	$6.89e^{-1}$	5228	0	0	12.98	$4.03e^{-1}$	$3.11e^{-2}$	3127.5	46.05	$1.47e^{-2}$
15×15	$6.78e^{-3}$	$6.13e^{-4}$	$9.03e^{-2}$	6058	0	0	16.70	$7.48e^{-1}$	$4.48e^{-2}$	3702.3	82.14	$2.22e^{-2}$
16×16	$8.50e^{-3}$	$5.37e^{-4}$	$6.32e^{-2}$	5675	0	0	21.96	1.03	$4.67e^{-2}$	4168	67.54	$1.62e^{-2}$
17×17	$1.00e^{-2}$	$7.83e^{-4}$	$7.81e^{-2}$	6247	0	0	28.01	1.68	$6.00e^{-2}$	4722.8	48.39	$1.03e^{-2}$
18×18	$1.37e^{-2}$	$3.87e^{-3}$	$2.83e^{-1}$	6436	0	0	35.26	3.89	$1.10e^{-1}$	5281.4	48.56	$9.19e^{-3}$
19×19	$1.73e^{-2}$	$8.84e^{-3}$	$5.12e^{-1}$	5264	0	0	42.78	3.18	$7.44e^{-2}$	4929.2	26.38	$5.35e^{-3}$
20×20	$1.94e^{-2}$	$5.48e^{-3}$	$2.82e^{-1}$	6697	0	0	48.76	2.92	$5.98e^{-2}$	6080.5	17.68	$2.91e^{-3}$
Avg.	-	-	$2.47e^{-1}$	-	-	0	-	-	$5.37e^{-2}$	-	-	$1.28e^{-2}$

TABLE XIV. COMPARISON OF COMPUTATIONAL TIME AND OBJECTIVE FUNCTION FOR BENCHMARK JSS INSTANCES

Instance	Greedy Algorithm						GRASP					
	Computational Times			Objective Function			Computational Time			Objective Function		
	$\overline{x}$	SD	CV	$\overline{x}$	SD	CV	$\overline{x}$	SD	CV	$\overline{x}$	SD	CV
abz5	$2.58e^{-3}$	$3.39e^{-3}$	1.31	7179	0	0	$5.97e^{-1}$	$3.37e^{-1}$	$5.64e^{-1}$	6123.3	136.57	$2.23e^{-2}$
abz6	$1.44e^{-3}$	$2.71e^{-3}$	1.88	7614	0	0	$5.66e^{-1}$	$1.24e^{-1}$	$2.18e^{-1}$	5067.7	168.55	$3.33e^{-2}$
ft10	$1.00e^{-3}$	$3.88e^{-4}$	$3.85e^{-1}$	6897	0	0	$6.19e^{-1}$	$1.36e^{-1}$	$2.19e^{-1}$	5538.3	177.82	$3.21e^{-2}$
orb01	$5.12e^{-4}$	$5.41e^{-4}$	1.06	8726	0	0	$6.19e^{-1}$	$5.13e^{-2}$	$8.29e^{-2}$	6507.6	300.82	$4.62e^{-2}$
orb02	$8.06e^{-4}$	$4.26e^{-4}$	$5.28e^{-1}$	5660	0	0	$5.87e^{-1}$	$1.16e^{-1}$	$1.98e^{-1}$	4288.3	240.90	$5.62e^{-2}$
orb03	$7.21e^{-4}$	$4.99e^{-4}$	$6.91e^{-1}$	9540	0	0	$5.20e^{-1}$	$1.14e^{-1}$	$2.20e^{-1}$	5707	363.37	$6.37e^{-2}$
orb04	$1.13e^{-3}$	$3.49e^{-4}$	$3.09e^{-1}$	8444	0	0	$9.63e^{-1}$	$1.81e^{-1}$	$1.88e^{-1}$	5104.1	301.49	$5.91e^{-2}$
orb05	$9.74e^{-4}$	$6.478e^{-4}$	$6.65e^{-1}$	11232	0	0	$6.73e^{-1}$	$2.20e^{-1}$	$3.27e^{-1}$	4823.4	406.65	$8.43e^{-2}$
ta01	$2.22e^{-3}$	$4.10e^{-4}$	$1.85e^{-1}$	74894	0	0	3.40	$5.73e^{-1}$	$1.69e^{-1}$	74182.8	253.81	$3.42e^{-3}$
ta02	$2.90e^{-3}$	$7.63e^{-4}$	$2.63e^{-1}$	55035	0	0	2.78	$3.57e^{-1}$	$1.28e^{-1}$	55035	0	0
ta03	$2.88e^{-3}$	$6.81e^{-4}$	$2.37e^{-1}$	73210	0	0	2.83	$5.41e^{-1}$	$1.91e^{-1}$	73210	0	0
Avg.	-	-	$6.82e^{-1}$	-	-	0	-	-	$2.28e^{-1}$	-	-	$3.64e^{-2}$

TABLE XV. COMPARISON OF COMPUTATIONAL TIME BETWEEN BENCHMARK JSS INSTANCES AND OUR STUDY

Benchmark instances	Previous studies	Avg. values from our study
ft06	0.37 [68], < 1 [114]	0.003
ft10	173.2 [68]	$6.19e^{-1}$
abz5	2.53 [56]	$5.97e^{-1}$
abz6	2.26 [56]	$5.66e^{-1}$
orb01	46.75 [56]	$6.19e^{-1}$
orb02	11.45 [56]	$5.87e^{-1}$
orb03	33.32 [56]	$5.20e^{-1}$
orb04	1.94 [56]	$9.63e^{-1}$
orb05	14.61 [56]	$6.73e^{-1}$

TABLE XVI. MACHINES AVAILABILITY (ATM) AND JOBS AVAILABILITY (JAT) FOR FT06 HYPOTHETICAL BIKE MANUFACTURING COMPANY AT T=15

M_i	Status	ATM	J_j	Status	JAT	Next machine	Earliest start time
M_1	Busy	18	J_1	Available	16	M_2	16
M_2	Busy	16	J_2	Busy	23	M_6	23
M_3	Busy	22	J_3	Busy	18	M_1	18
M_4	Busy	15	J_4	Busy	22	M_3	22
M_5	Busy	23	J_5	Busy	22	M_2	22
M_6	Busy	17	J_6	Busy	16	M_4	16

TABLE XVII. COMPARISON OF DIFFERENT RESCHEDULING METHODS FOR A HYPOTHETICAL FT06 BIKE MANUFACTURING COMPANY

Rescheduling Strategy	Rescheduling method	Rescheduling policy	Dynamic event	Total tardiness before rescheduling	Total tardiness after rescheduling
Reactive	Regenerative	Event-driven	2-rush orders	32	33
Reactive	Right shift	Event-driven	2- Machine fails	32	38
Proactive	Regenerative	Periodic	2- new jobs arrival	32	38
Proactive	Regenerative	Periodic	Scheduled machine maintenance	32	43

VIII. CONCLUSION

This study addressed the complex NP-hard JSS problem, focusing on the challenges posed by serious dynamic events (SDE) such as new job arrivals (NJA), rush orders (RO), machine failures (MF) and scheduled machine maintenance (SMM). The research objective was to develop and compare the exact methods, a Greedy Algorithm (G_rA), and a novel Greedy Randomized Adaptive Search Procedure (GRASP) to efficiently solve JSS problems under various conditions. The study contributed by developing an exact method for benchmarking, G_rA for faster solutions and a novel GRASP algorithm featuring a directed operations swapping procedure to achieve high-quality solutions with computational efficiency. Additionally, a proactive-reactive ($P_{act} - R_{act}$) rescheduling strategy in handling SDE such as NJA, RO, SMM, and MF using a right shift (RF) and regeneration (Reg) methods at hybrid (Hyb) policy have been implemented. The main findings of the study are:

a) The exact method: it provided optimal benchmarks for small to medium instances, ranging from 2×2 to 9×9, while demonstrating exponential computational growth, reaching 23758.62 sec (approximately 6.6 hours) for a 9×9 instance. Exact method showed limited applicability to medium to large instances due to computational complexity, for example, the optimal solution for 10×10 instance could not be obtained even after 48 hours.

b) G_rA: $\mathrm{G_rA}$ offered faster solutions compared to the exact method for small to medium instances. For 9×9 instance, $\mathrm{G_rA}$'s computational time was only $1.34\,\mathrm{e}^{-3}$ sec which was 23758.62 sec for the same instance using the exact method. Additionally, $\mathrm{G_rA}$ showed a higher average percentage gap of 13.82% for small to medium instances when compared with the exact methods. Because of this reason, it has a limited suitability for scenarios requiring high solution quality. However, the $\mathrm{G_rA}$ demonstrated significant computational efficiency for medium to large and benchmark instances. For example, it achieved computational times of $1.96\mathrm{e}^{-3}$ for 10×10 and $1.94\ \mathrm{e}^{-2}$ for 20×20. This efficiency was also evident in benchmark instances, where computational times were consistently in the order of e^{-3}. These results suggest that the $\mathrm{G_rA}$ might be more suitable for applications where quick approximate solutions are preferred over optimal.

c) GRASP: GRASP consistently outperformed both exact and $\mathrm{G_rA}$ approaches with average percentage gap of only 3.43% for small to medium instances, compared to $\mathrm{G_rA}$'s 13.82%. It also demonstrated superior performance for medium to large instances. For example, for 10×10, GRASP reduced the total tardiness value from 1937 in $\mathrm{G_rA}$ to 1296.2, and for 20×20 instances, from 6697 to 6080.5, while maintaining reasonable computational time. Similarly, for benchmark instances, GRASP showed significant improvements. In the case of abz05, the objective function value was reduced from 7179 using $\mathrm{G_rA}$ to 6123.3 with GRASP, demonstrating 14.7% improvement for this instance. This improvement was consistent among all other benchmark instances. Moreover, GRASP achieved remarkable reductions in computational time for benchmark instances from literature: for the ft06 instance, the time decreased from 0.37 seconds in previous studies to 0.003 seconds, and for the ft10 instance, from 173.2 seconds to 0.619 seconds. These results highlight GRASP's ability to consistently yield high-quality solutions efficiently across various instance sizes, from small to large, as well as for established benchmark problems. The research also demonstrated the effectiveness of the proactive-reactive ($\mathrm{P_{act}} - \mathrm{R_{act}}$) rescheduling strategy in handling SDE such as NJA, RO, SMM, and MF.

While the study presented promising results, it is limited using historical data, a single objective function, and the consideration of only SDE. Future research could address these limitations:

(1) By incorporating a wider range of dynamic events, including medium and more serious events. These dynamic events lead to continuous system updates, causing nervousness, deviations from the original schedule, and reduced execution efficiency.

(2) By empirical validation using real production data that could further enhance the practical relevance and implementation viability of the proposed methods.

(3) By exploring the performance of the JSS under multi objective functions, such as makespan or flow time

Furthermore, to provide monitoring, control, and prediction capabilities for JSS amidst dynamic events, JSS could be integrated with Digital Twin (DT) technology. DT has the potential to detect these dynamic events in real-time, offering immediate insights and enabling rapid responses.

DT technology bridges the physical and virtual JSS environments by enabling real-time mapping and bidirectional interaction. This integration allows for the collection of detailed real-time information on machines, jobs, operations, equipment, inventory, and work-in-progress (WIP). By combining operational data, environmental changes, and dynamic events from the physical JSS with virtual data, DT technology facilitates real-time information flow. It enhances rescheduling by continuously comparing physical JSS data with its virtual counterpart to detect events in real time. When an event occurs, DT triggers a rescheduling policy, sending updated data to the virtual JSS. The virtual JSS then reschedules the unfinished operations using built-in algorithms and provides the optimal schedule back to the physical JSS's Manufacturing Execution System (MES), enabling dynamic scheduling and reducing deviations between planned and actual schedules.

However, the practical implementation of integrating JSS with DT presents several challenges:

1) The need for data acquisition and processing capabilities: real-time data from IoT sensors must be accurately collected, processed, and transmitted to the DT. The study by [118] could be used for this purpose, who developed a prototype for integrating sensors with IoT to track system components to obtain data. They used the collected data to update the production schedule within an ERP system. This prototype could be used to extend our research to enable real-time monitoring capabilities.

2) The integration of cyber and physical world with JSS: utilizing Industry 4.0 concepts such as CPS can greatly enhance the capability of production systems to be managed, monitored, and controlled, ultimately improving production scheduling, and rapid response, in industrial processes.

Our research has practicality and real-world relevance in automotive manufacturing, electronics manufacturing, and other companies. The findings highlight the potential of our approach to enhance efficiency, highlighting the need for continued research in this area.

References

[1] H. Numaguchi, W. Wu, and Y. Hu, "Two-machine job-shop scheduling with one joint job," *Discrete Appl Math (1979)*, vol. 346, pp. 30–43, Mar. 2024, doi: 10.1016/j.dam.2023.11.037.

[2] K. Tamssaouet and S. Dauzère-Pérès, "A general efficient neighborhood structure framework for the job-shop and flexible job-shop scheduling problems," *Eur J Oper Res*, vol. 311, no. 2, 2023, doi: 10.1016/j.ejor.2023.05.018.

[3] M. Đurasević and D. Jakobović, "A survey of dispatching rules for the dynamic unrelated machines environment," *Expert Syst Appl*, vol. 113, 2018, doi: 10.1016/j.eswa.2018.06.053.

[4] M. Ortíz-Barrios, A. Petrillo, F. De Felice, N. Jaramillo-Rueda, G. Jiménez-Delgado, and L. Borrero-López, "A dispatching-fuzzy ahp-topsis model for scheduling flexible job-shop systems in industry 4.0 context," *Applied Sciences*, vol. 11, no. 11, 2021, doi: 10.3390/app11115107.

[5] M. L. Pinedo. *Scheduling: Theory, Algorithms, and Systems, Sixth Edition*. Springer Cham, 2022, doi: 10.1007/978-3-031-05921-6.

[6] D. Karunakaran. *Active Learning Methods for Dynamic Job Shop Scheduling using Genetic Programming under Uncertain Environment*. Doctoral dissertation, Open Access Te Herenga Waka-Victoria University of Wellington, 2019.

[7] K. Tliba, T. M. L. Diallo, O. Penas, R. Ben Khalifa, N. Ben Yahia, and J. Y. Choley, "Digital twin-driven dynamic scheduling of a hybrid flow shop," *J Intell Manuf*, vol. 34, no. 5, 2023, doi: 10.1007/s10845-022-01922-3.

[8] D. Wang, Y. Yin, and Y. Jin. *Rescheduling Under Disruptions in Manufacturing Systems: Models and Algorithms*. Uncertainty and Operations Research, 2020.

[9] M. R. Singh and R. Mishra. *A study on flexible flow shop and job shop scheduling using meta-heuristic approaches*. Doctoral dissertation, National Institute of Technology, Rourkela, 2014.

[10] M. Saqlain, S. Ali, and J. Y. Lee, "A Monte-Carlo tree search algorithm for the flexible job-shop scheduling in manufacturing systems," *Flex Serv Manuf J*, vol. 35, no. 2, pp. 548-571, 2022, doi: 10.1007/s10696-021-09437-4.

[11] F. Zhang, Y. Mei, and M. Zhang, "A new representation in genetic programming for evolving dispatching rules for dynamic flexible job shop scheduling," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, pp. 33-49, 2019, doi: 10.1007/978-3-030-16711-0_3.

[12] S. R. Kamali, T. Banirostam, H. Motameni, and M. Teshnehlab, "An immune-based multi-agent system for flexible job shop scheduling problem in dynamic and multi-objective environments," *Eng Appl Artif Intell*, vol. 123, 2023, doi: 10.1016/j.engappai.2023.106317.

[13] S. Shahbazi, S. M. Sajadi, and F. Jolai, "A Simulation-Based Optimization Model for Scheduling New Product Development Projects in Research and Development Centers," *Iranian Journal of Management Studies*, vol. 10, no. 4, 2017.

[14] G. Da Col and E. C. Teppan, "Industrial-size job shop scheduling with constraint programming," *Operations Research Perspectives*, vol. 9, 2022, doi: 10.1016/j.orp.2022.100249.

[15] H. Wang, J. Cheng, C. Liu, Y. Zhang, S. Hu, and L. Chen, "Multi-objective reinforcement learning framework for dynamic flexible job shop scheduling problem with uncertain events," *Appl Soft Comput*, vol. 131, 2022, doi: 10.1016/j.asoc.2022.109717.

[16] E. Nawara and G. Hassanein, "Solving the job-shop scheduling problem by arena simulation software," *International Journal of Engineering Innovation & Research*, vol. 2, no. 2, 2013.

[17] S. Afsar, C. R. Vela, J. J. Palacios, and I. González-Rodríguez, “Mathematical models and benchmarking for the fuzzy job shop scheduling problem,” *Comput Ind Eng*, vol. 183, 2023, doi: 10.1016/j.cie.2023.109454.

[18] K. Kurowski, T. Pecyna, M. Slysz, R. Różycki, G. Waligóra, and J. Węglarz, “Application of quantum approximate optimization algorithm to job shop scheduling problem,” *Eur J Oper Res*, vol. 310, no. 2, 2023, doi: 10.1016/j.ejor.2023.03.013.

[19] M. Vali, K. Salimifard, A. H. Gandomi, and T. J. Chaussalet, “Application of job shop scheduling approach in green patient flow optimization using a hybrid swarm intelligence,” *Comput Ind Eng*, vol. 172, 2022, doi: 10.1016/j.cie.2022.108603.

[20] R. Zhang, S. Song, and C. Wu, “A hybrid artificial bee colony algorithm for the job shop scheduling problem,” *Int J Prod Econ*, vol. 141, no. 1, 2013, doi: 10.1016/j.ijpe.2012.03.035.

[21] M. M. Gohareh and E. Mansouri, “A simulation-optimization framework for generating dynamic dispatching rules for stochastic job shop with earliness and tardiness penalties,” *Comput Oper Res*, vol. 140, 2022, doi: 10.1016/j.cor.2021.105650.

[22] H. Xiong, S. Shi, D. Ren, and J. Hu, “A survey of job shop scheduling problem: The types and models,” *Computers and Operations Research*, vol. 142. 2022. doi: 10.1016/j.cor.2022.105731.

[23] Y. Fang, C. Peng, P. Lou, Z. Zhou, J. Hu, and J. Yan, “Digital-Twin-Based Job Shop Scheduling Toward Smart Manufacturing,” *IEEE Trans Industr Inform*, vol. 15, no. 12, 2019, doi: 10.1109/TII.2019.2938572.

[24] S. Habbadi, B. Herrou, and S. Sekkat, “Job Shop Scheduling Problem Using Genetic Algorithms,” *5th European International Conference on Industrial Engineering and Operations Management*, 2023, doi: 10.46254/eu05.20220592.

[25] F. M. Defersha, D. Obimuyiwa, and A. D. Yimer, “Mathematical model and simulated annealing algorithm for setup operator constrained flexible job shop scheduling problem,” *Comput Ind Eng*, vol. 171, 2022, doi: 10.1016/j.cie.2022.108487.

[26] H. Nazif, “An effective meta-heuristic algorithm to minimize makespan in job shop scheduling,” *Industrial Engineering and Management Systems*, vol. 18, no. 3, 2019, doi: 10.7232/iems.2019.18.3.360.

[27] B. Firme, J. Figueiredo, J. M. C. Sousa, and S. M. Vieira, “Agent-based hybrid tabu-search heuristic for dynamic scheduling,” *Eng Appl Artif Intell*, vol. 126, 2023, doi: 10.1016/j.engappai.2023.107146.

[28] M. Zhang, F. Tao, and A. Y. C. Nee, “Digital Twin Enhanced Dynamic Job-Shop Scheduling,” *J Manuf Syst*, vol. 58, 2021, doi: 10.1016/j.jmsy.2020.04.008.

[29] L. Liu, K. Guo, Z. Gao, J. Li, and J. Sun, “Digital Twin-Driven Adaptive Scheduling for Flexible Job Shops,” *Sustainability (Switzerland)*, vol. 14, no. 9, 2022, doi: 10.3390/su14095340.

[30] M. Ghaleb, S. Taghipour, and H. Zolfagharinia, “Real-time integrated production-scheduling and maintenance-planning in a flexible job shop with machine deterioration and condition-based maintenance,” *J Manuf Syst*, vol. 61, 2021, doi: 10.1016/j.jmsy.2021.09.018.

[31] E. Teppan, “Types of Flexible Job Shop Scheduling: A Constraint Programming Experiment,” in *ICAART*, no. 3, pp. 516-523, 2022, doi: 10.5220/0010849900003116.

[32] K. Li, Q. Deng, L. Zhang, Q. Fan, G. Gong, and S. Ding, “An effective MCTS-based algorithm for minimizing makespan in dynamic flexible job shop scheduling problem,” *Comput Ind Eng*, vol. 155, 2021, doi: 10.1016/j.cie.2021.107211.

[33] Z. Zhuang, Y. Li, Y. Sun, W. Qin, and Z. H. Sun, “Network-based dynamic dispatching rule generation mechanism for real-time production scheduling problems with dynamic job arrivals,” *Robot Comput Integr Manuf*, vol. 73, 2022, doi: 10.1016/j.rcim.2021.102261.

[34] S. Tian, T. Wang, L. Zhang, and X. Wu, “Real-time shop floor scheduling method based on virtual queue adaptive control: Algorithm and experimental results,” *Measurement*, vol. 147, 2019, doi: 10.1016/j.measurement.2019.05.080.

[35] A. Tighazoui, C. Sauvey, and N. Sauer, “Predictive-reactive strategy for identical parallel machine rescheduling,” *Comput Oper Res*, vol. 134, 2021, doi: 10.1016/j.cor.2021.105372.

[36] L. J. Song, H. P. Gu, S. Y. Jin, and H. Zhao, “Rescheduling methods for manufacturing firms applying make-to-order strategy,” *International Journal of Simulation Modelling*, vol. 14, no. 4, 2015.

[37] G. E. Vieira, J. W. Herrmann, and E. Lin, “Rescheduling manufacturing systems: A framework of strategies, policies, and methods,” in *Journal of Scheduling*, vol. 6, pp. 39-62, 2003, doi: 10.1023/A:1022235519958.

[38] D. Ouelhadj and S. Petrovic, “A survey of dynamic scheduling in manufacturing systems,” *Journal of Scheduling*, vol. 12, no. 4. 2009. doi: 10.1007/s10951-008-0090-8.

[39] L. Mönch, J. W. Fowler, and S. J. Mason. *Production Planning and Control for Semiconductor Wafer Fabrication Facilities: Modeling, Analysis, and Systems, vol. 52*. Springer Science & Business Media, 2013.

[40] A. K. Jain and H. A. Elmaraghy, “Production scheduling/rescheduling in flexible manufacturing,” *Int J Prod Res*, vol. 35, no. 1, 1997, doi: 10.1080/002075497196082.

[41] L. Zhang, L. Gao, and X. Li, “A hybrid intelligent algorithm and rescheduling technique for job shop scheduling problems with disruptions,” *International Journal of Advanced Manufacturing Technology*, vol. 65, no. 5–8, 2013, doi: 10.1007/s00170-012-4245-6.

[42] M. Wang, P. Zhang, P. Zheng, J. He, J. Zhang, and J. Bao, “An Improved Genetic Algorithm with Local Search for Dynamic Job Shop Scheduling Problem,” in *IEEE International Conference on Automation Science and Engineering*, pp. 766-771, 2020, doi: 10.1109/CASE48305.2020.9216737.

[43] M. A. Aloulou and M.-C. Portmann, “An Efficient Proactive-Reactive Scheduling Approach to Hedge Against Shop Floor Disturbances,” in *Multidisciplinary Scheduling: Theory and Applications*, pp. 223-246, 2005, doi: 10.1007/0-387-27744-7_11.

[44] D. Rahmani, M. Heydari, A. Makui, and M. Zandieh, “A new approach to reducing the effects of stochastic disruptions in flexible flow shop problems with stability and nervousness,” *International Journal of Management Science and Engineering Management*, vol. 8, no. 3, 2013, doi: 10.1080/17509653.2013.812332.

[45] Z. Yahouni, N. Mebarki, and Z. Sari, “Evaluation of a new decision-aid parameter for job shop scheduling under uncertainties,” *RAIRO - Operations Research*, vol. 53, no. 2, 2019, doi: 10.1051/ro/2017073.

[46] V. Rahimi, J. Arkat, and H. Farughi, “Reactive scheduling addressing unexpected disturbance in cellular manufacturing systems,” *International Journal of Engineering, Transactions A: Basics*, vol. 34, no. 1, 2021, doi: 10.5829/IJE.2021.34.01A.18.

[47] S. Fatemi-Anaraki, R. Tavakkoli-Moghaddam, M. Foumani, and B. Vahedi-Nouri, “Scheduling of Multi-Robot Job Shop Systems in Dynamic Environments: Mixed-Integer Linear Programming and Constraint Programming Approaches,” *Omega (United Kingdom)*, vol. 115, 2023, doi: 10.1016/j.omega.2022.102770.

[48] J. Adams, E. Balas, and D. Zawack, “Shifting Bottleneck Procedure For Job Shop Scheduling.,” *Manage Sci*, vol. 34, no. 3, 1988, doi: 10.1287/mnsc.34.3.391.

[49] S. Mahmud, A. Abbasi, R. K. Chakrabortty, and M. J. Ryan, “Multi-operator communication based differential evolution with sequential Tabu Search approach for job shop scheduling problems,” *Appl Soft Comput*, vol. 108, 2021, doi: 10.1016/j.asoc.2021.107470.

[50] E. J. Kontoghiorghes. *Handbook of parallel computing and statistics*. CRC Press, 2005, doi: 10.1198/tech.2008.s912.

[51] M. Rajkumar, P. Asokan, N. Anilkumar, and T. Page, “A GRASP algorithm for flexible job-shop scheduling problem with limited resource constraints,” *Int J Prod Res*, vol. 49, no. 8, 2011, doi: 10.1080/00207541003709544.

[52] P. Festa and M. G. C. Resende, “An annotated bibliography of GRASP – Part I: Algorithms,” *International Transactions in Operational Research*, vol. 16, no. 1, 2009, doi: 10.1111/j.1475-3995.2009.00663.x.

[53] S. R. Gupta and J. S. Smith, “Algorithms for single machine total tardiness scheduling with sequence dependent setups,” *Eur J Oper Res*, vol. 175, no. 2, 2006, doi: 10.1016/j.ejor.2005.05.018.

[54] A. Corberán, R. Martí, and J. M. Sanchis, “A GRASP heuristic for the mixed Chinese postman problem,” *Eur J Oper Res*, vol. 142, no. 1, 2002, doi: 10.1016/S0377-2217(01)00296-X.

[55] G. Prabhaharan, B. S. H. Khan, and L. Rakesh, "Implementation of grasp in flow shop scheduling," *International Journal of Advanced Manufacturing Technology*, vol. 30, no. 11–12, 2006, doi: 10.1007/s00170-005-0134-6.

[56] R. M. Aiex, S. Binato, and M. G. C. Resende, "Parallel GRASP with path-relinking for job shop scheduling," in *Parallel Computing*, vol. 29, no. 4, pp. 393-430, 2003. doi: 10.1016/S0167-8191(03)00014-0.

[57] K. Morikawa, K. Nagasawa, and K. Takahashi, "Job shop scheduling by branch and bound using genetic programming," in *Procedia Manufacturing*, vol. 39, 1112-1118, 2019.

[58] L. Zhang, Y. Hu, C. Wang, Q. Tang, and X. Li, "Effective dispatching rules mining based on near-optimal schedules in intelligent job shop environment," *J Manuf Syst*, vol. 63, 2022, doi: 10.1016/j.jmsy.2022.04.019.

[59] M. Paul, R. Sridharan, and T. R. Ramanan, "Scheduling of an assembly job shop: A case study based on hydraulic manufacturing industry," in *Materials Today: Proceedings*, vol. 47, pp. 4988-4992, 2021, doi: 10.1016/j.matpr.2021.04.341.

[60] C. H. Akarsu and T. Küçükdeniz, "Job shop scheduling with genetic algorithm-based hyperheuristic approach," *International Advanced Researches and Engineering Journal*, vol. 6, no. 1, 2022, doi: 10.35860/iarej.1018604.

[61] S. Chakraborty and S. Bhowmik, "Job Shop Scheduling using Simulated Annealing," *Hooghly Engineering & Technology College*, vol. 1, no. 1, pp. 69-73, 2013.

[62] M. H. Ali, A. Saif, and A. Ghasemi, "Robust Job Shop Scheduling with Condition-Based Maintenance and Random Breakdowns," in *IFAC-PapersOnLine*, vol. 55, no. 10, pp. 1225-1230, 2022, doi: 10.1016/j.ifacol.2022.09.557.

[63] K. S. Sundari, "Makespan Minimization in Job Shop Scheduling," *International Journal of Engineering and Management Research*, vol. 11, no. 1, 2021, doi: 10.31033/ijemr.11.1.31.

[64] J. F. Bard and T. A. Feo, "Note—Operations Sequencing in Discrete Parts Manufacturing," *Manage Sci*, vol. 35, no. 2, 1989, doi: 10.1287/mnsc.35.2.249.

[65] H. Akrout, B. Jarboui, A. Rebaï, and P. Siarry, "New Greedy Randomized Adaptive Search Procedure based on differential evolution algorithm for solving no-wait flowshop scheduling problem," in *2013 International Conference on Advanced Logistics and Transport, ICALT 2013*, pp. 327-334, 2013, doi: 10.1109/ICAdLT.2013.6568480.

[66] A. Sayah, S. Aqil, and M. Lahby, "Minimizing Maximum Tardiness in a Distributed Flow Shop Manufacturing Problem under No-Waiting and Sequence Dependent Setup Time Constraints," in *Proceedings - SITA 2023: 2023 14th International Conference on Intelligent Systems: Theories and Applications*, pp. 1-6, 2023, doi: 10.1109/SITA60746.2023.10373688.

[67] M. Laguna and J. L. G. Velarde, "A search heuristic for just-in-time scheduling in parallel machines," *J Intell Manuf*, vol. 2, no. 4, 1991, doi: 10.1007/BF01471113.

[68] A. Baykasoğlu and F. S. Karaslan, "Solving comprehensive dynamic job shop scheduling problem by using a GRASP-based approach," *Int J Prod Res*, vol. 55, no. 11, 2017, doi: 10.1080/00207543.2017.1306134.

[69] T. Witkowski, P. Antczak, and A. Antczak, "Solving the flexible open-job shop scheduling problem with GRASP and Simulated Annealing," in *Proceedings - International Conference on Artificial Intelligence and Computational Intelligence, AICI 2010*, vol. 2, pp. 437-442, 2010, doi: 10.1109/AICI.2010.212.

[70] A. Baykasoğlu and F. S. Madenoğlu, "Greedy randomized adaptive search procedure for simultaneous scheduling of production and preventive maintenance activities in dynamic flexible job shops," *Soft comput*, vol. 25, no. 23, 2021, doi: 10.1007/s00500-021-06053-0.

[71] T. Witkowski, A. Antczak, and P. Antczak, "Using GRASP for optimization of flow production in FJSP problem with transportation operations," in *Proceedings - International Conference on Natural Computation*, pp. 1255-1261, 2012.

[72] M. Essafi, X. Delorme, and A. Dolgui, "A GRASP heuristic for sequence-dependent transfer line balancing problem," in *IFAC Proceedings Volumes (IFAC-PapersOnline)*, vol. 42, no. 4, pp. 762-767, 2009, doi: 10.3182/20090603-3-RU-2001.0500.

[73] A. N. Júnior and L. R. Guimarães, "A greedy randomized adaptive search procedure application to solve the travelling salesman problem," *International Journal of Industrial Engineering and Management*, vol. 10, no. 3, 2019, doi: 10.24867/IJIEM-2019-3-243.

[74] Z. Liu *et al.*, "A graph neural networks-based deep Q-learning approach for job shop scheduling problems in traffic management," *Inf Sci (N Y)*, vol. 607, 2022, doi: 10.1016/j.ins.2022.06.017.

[75] J. B. Atkinson, "A greedy randomised search heuristic for time constrained vehicle scheduling and the incorporation of a learning strategy," *Journal of the Operational Research Society*, vol. 49, no. 7, 1998, doi: 10.1057/palgrave.jors.2600521.

[76] J. Xie, X. Li, L. Gao, and L. Gui, "A hybrid genetic tabu search algorithm for distributed flexible job shop scheduling problems," *J Manuf Syst*, vol. 71, 2023, doi: 10.1016/j.jmsy.2023.09.002.

[77] W. Y. Ku and J. C. Beck, "Mixed Integer Programming models for job shop scheduling: A computational analysis," *Comput Oper Res*, vol. 73, 2016, doi: 10.1016/j.cor.2016.04.006.

[78] S. Meeran and M. S. Morshed, "A hybrid genetic tabu search algorithm for solving job shop scheduling problems: A case study," *Journal of Intelligent Manufacturing*, vol. 23, no. 4. 2012. doi: 10.1007/s10845-011-0520-x.

[79] M. A. Salido, J. Escamilla, A. Giret, and F. Barber, "A genetic algorithm for energy-efficiency in job-shop scheduling," *International Journal of Advanced Manufacturing Technology*, vol. 85, no. 5–8, 2016, doi: 10.1007/s00170-015-7987-0.

[80] R. Zhang and R. Chiong, "Solving the energy-efficient job shop scheduling problem: A multi-objective genetic algorithm with enhanced local search for minimizing the total weighted tardiness and total energy consumption," *J Clean Prod*, vol. 112, 2016, doi: 10.1016/j.jclepro.2015.09.097.

[81] J. Zhu, Z. H. Shao, and C. Chen, "An improved whale optimization algorithm for job-shop scheduling based on quantum computing," *International Journal of Simulation Modelling*, vol. 18, no. 3, 2019, doi: 10.2507/IJSIMM18(3)CO13.

[82] Z. Zhang, Z. L. Guan, J. Zhang, and X. Xie, "A novel job-shop scheduling strategy based on particle swarm optimization and neural network," *International Journal of Simulation Modelling*, vol. 18, no. 4, 2019, doi: 10.2507/IJSIMM18(4)CO18.

[83] P. Lou, Q. Liu, Z. Zhou, H. Wang, and S. X. Sun, "Multi-agent-based proactive-reactive scheduling for a job shop," *International Journal of Advanced Manufacturing Technology*, vol. 59, no. 1–4, 2012, doi: 10.1007/s00170-011-3482-4.

[84] I. Paprocka, "Evaluation of the effects of a machine failure on the robustness of a job shop system-proactive approaches," *Sustainability (Switzerland)*, vol. 11, no. 1, 2019, doi: 10.3390/su11010065.

[85] J. Fang and Y. Xi, "A rolling horizon job shop rescheduling strategy in the dynamic environment," *International Journal of Advanced Manufacturing Technology*, vol. 13, no. 3, 1997, doi: 10.1007/BF01305874.

[86] M. A. Adibi, M. Zandieh, and M. Amiri, "Multi-objective scheduling of dynamic job shop using variable neighborhood search," *Expert Syst Appl*, vol. 37, no. 1, 2010, doi: 10.1016/j.eswa.2009.05.001.

[87] Z. Li, "Multi-task scheduling optimization in shop floor based on uncertainty theory algorithm," *Academic Journal of Manufacturing Engineering*, vol. 17, no. 1, 2019.

[88] F. Echsler Minguillon and N. Stricker, "Robust predictive–reactive scheduling and its effect on machine disturbance mitigation," *CIRP Annals*, vol. 69, no. 1, 2020, doi: 10.1016/j.cirp.2020.03.019.

[89] S. Luo, L. Zhang, and Y. Fan, "Dynamic multi-objective scheduling for flexible job shop by deep reinforcement learning," *Comput Ind Eng*, vol. 159, 2021, doi: 10.1016/j.cie.2021.107489.

[90] L. Liu, "Outsourcing and rescheduling for a two-machine flow shop with the disruption of new arriving jobs: A hybrid variable neighborhood search algorithm," *Comput Ind Eng*, vol. 130, 2019, doi: 10.1016/j.cie.2019.02.015.

[91] H. Aytug, M. A. Lawley, K. McKay, S. Mohan, and R. Uzsoy, "Executing production schedules in the face of uncertainties: A review and some future directions," in *European Journal of Operational Research*, vol. 161, no. 1, pp. 86-110, 2005, doi: 10.1016/j.ejor.2003.08.027.

[92] Y. F. Wang, Y. F. Zhang, J. Y. H. Fuh, Z. D. Zhou, P. Lou, and L. G. Xue, “An integrated approach to reactive scheduling subject to machine breakdown,” in *Proceedings of the IEEE International Conference on Automation and Logistics, ICAL 2008*, pp. 542-547, 2008, doi: 10.1109/ICAL.2008.4636210.

[93] J. F. Jimenez, E. Gonzalez-Neira, and G. Zambrano-Rey, “An adaptive genetic algorithm for a dynamic single-machine scheduling problem,” *Management Science Letters*, vol. 8, no. 11, 2018, doi: 10.5267/j.msl.2018.8.011.

[94] B. Yang, “Single machine rescheduling with new jobs arrivals and processing time compression,” *International Journal of Advanced Manufacturing Technology*, vol. 34, no. 3–4, 2007, doi: 10.1007/s00170-006-0590-7.

[95] K. Muhamadin *et al.*, “A Review for Dynamic Scheduling in Manufacturing,” *Type: Double Blind Peer Reviewed International Research Journal Publisher: Global Journals Online*, vol. 18, no. 5, 2018.

[96] K. Murakami and H. Morita, “A Method for Generating Robust Schedule under Uncertainty in Processing Time (< Special Issue> TOTAL OPERATIONS MANAGEMENT),” *International Journal of Biomedical Soft Computing and Human Sciences: the official journal of the Biomedical Fuzzy Systems Association*, vol. 15, no. 1, pp. 45-50, 2010.

[97] L. H. Wu, X. Chen, X. D. Chen, and Q. X. Chen, “The research on proactive-reactive scheduling framework based on real-time manufacturing information,” in *Materials Science Forum*, vol. 626, pp. 789-794, 2009.

[98] X. Wen, X. Lian, Y. Qian, Y. Zhang, H. Wang, and H. Li, “Dynamic scheduling method for integrated process planning and scheduling problem with machine fault,” *Robot Comput Integr Manuf*, vol. 77, 2022, doi: 10.1016/j.rcim.2022.102334.

[99] M. M. Tawfeek, Y. M. Sadek, and A. M. K. El-kharbotly, “Study of event-driven and periodic rescheduling on a single machine with unexpected disruptions,” *Independent Journal of Management & Production*, vol. 10, no. 1, 2019, doi: 10.14807/ijmp.v10i1.838.

[100] L. K. Church and R. Uzsoy, “Analysis of periodic and event-driven rescheduling policies in dynamic shops,” *Int J Comput Integr Manuf*, vol. 5, no. 3, pp. 153–163, 1992, doi: 10.1080/09511929208944524.

[101] G. E. Vieira, J. W. Herrmann, and E. Lin, “Analytical models to predict the performance of a single-machine system under periodic and event-driven rescheduling strategies,” *Int J Prod Res*, vol. 38, no. 8, 2000, doi: 10.1080/002075400188654.

[102] Y. Gao, Y. S. Ding, and H. Y. Zhang, “Job-shop scheduling considering rescheduling in uncertain dynamic environment,” in *2009 International Conference on Management Science and Engineering - 16th Annual Conference Proceedings, ICMSE 2009*, pp. 380-384, 2009, doi: 10.1109/ICMSE.2009.5317409.

[103] R. Barták and M. Vlk, “Reactive recovery from machine breakdown in production scheduling with temporal distance and resource constraints,” in *ICAART 2015 - 7th International Conference on Agents and Artificial Intelligence, Proceedings*, vol. 2, pp. 119-130, 2015, doi: 10.5220/0005215701190130.

[104] Y. Sang, J. Tan, and W. Liu, “A new many-objective green dynamic scheduling disruption management approach for machining workshop based on green manufacturing,” in *Journal of Cleaner Production*, vol. 297, 126489, 2021, doi: 10.1016/j.jclepro.2021.126489.

[105] A. Tighazoui, C. Sauvey, and N. Sauer, “New efficiency-stability criterion in a rescheduling problem with dynamic jobs weights,” in *7th International Conference on Control, Decision and Information Technologies, CoDIT 2020*, vol. 1, pp. 475-480, 2020, doi: 10.1109/CoDIT49905.2020.9263807.

[106] A. S. Muhamad and S. Deris, “Rescheduling for JSSP and FJSSP using Clonal Selection Principle Approach–A Theory,” *Journal Information And Technology Management (JISTM)*, vol. 1, no. 1, pp. 10-17, 2016.

[107] H. Fisher and G. L. Thompson, “Probabilistic Learning Combinations of Local Job-Shop Scheduling Rules,” in *Industrial Scheduling*, 1963.

[108] W. Hassanein, G. M. Nawara, and E. S. Wael Hassanein, “Solving the Job-Shop Scheduling Problem by Arena Simulation Software Productivity View project Solving the Job-Shop Scheduling Problem by Arena Simulation Software,” *International Journal of Engineering Innovations and Research*, vol. 2, no. 2, p. 161, 2014.

[109] E. G. Talbi. *Metaheuristics: From Design to Implementation*. John Wiley & Sons, 2009, doi: 10.1002/9780470496916.

[110] J. Zhang, G. Ding, Y. Zou, S. Qin, and J. Fu, “Review of job shop scheduling research and its new perspectives under Industry 4.0,” *J Intell Manuf*, vol. 30, no. 4, 2019, doi: 10.1007/s10845-017-1350-2.

[111] R. Martí, P. M. Pardalos, and M. G. C. Resende. *Handbook of heuristics*. Springer Publishing Company, Incorporated, 2018, doi: 10.1007/978-3-319-07124-4.

[112] K. R. Baker and D. Trietsch. *Principles of Sequencing and Scheduling*. John Wiley & Sons, 2018, doi: 10.1002/9780470451793.

[113] S. Strassl and N. Musliu, “Instance space analysis and algorithm selection for the job shop scheduling problem,” *Comput Oper Res*, vol. 141, 2022, doi: 10.1016/j.cor.2021.105661.

[114] J. A. S. Gromicho, J. J. Van Hoorn, F. Saldanha-Da-Gama, and G. T. Timmer, “Solving the job-shop scheduling problem optimally by dynamic programming,” *Comput Oper Res*, vol. 39, no. 12, 2012, doi: 10.1016/j.cor.2012.02.024.

[115] E. Taillard, “Benchmarks for basic scheduling problems,” *Eur J Oper Res*, vol. 64, no. 2, 1993, doi: 10.1016/0377-2217(93)90182-M.

[116] D. Applegate and W. Cook, “Computational study of the job-shop scheduling problem,” *ORSA journal on computing*, vol. 3, no. 2, 1991, doi: 10.1287/ijoc.3.2.149.

[117] L. Duan and B. Eng. *Applying Systematic Local Search To Job Shop Scheduling Problems*. Doctoral dissertation, Simon Fraser University, 2006.

[118] S. C. Adisasmito, P. D. Pamungkas, and A. Ma’Ruf, “Real-time monitoring design for make-to-order industry,” in *AIP Conference Proceedings*, vol. 2470, no. 1, 2022, doi: 10.1063/5.0080747.

[119] J. M. Framinan, V. Fernandez-Viagas, and P. Perez-Gonzalez, “Using real-time information to reschedule jobs in a flowshop with variable processing times,” *Comput Ind Eng*, vol. 129, 2019, doi: 10.1016/j.cie.2019.01.036.