IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 26 June 2023, accepted 7 July 2023, date of publication 12 July 2023, date of current version 26 July 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3294573

RESEARCH ARTICLE

Rapid Production Rescheduling for Flow Shop Under Machine Failure Disturbance Using Hybrid Perturbation Population Genetic Algorithm-Artificial Neural Networks (PPGA-ANNs)

PAKKAPORN SAOPHAN[1], WARUT PANNAKKONG[2], RAVEEKIAT SINGHAPHANDU[1,2], AND VAN-NAM HUYNH[1], (Member, IEEE)

[1]School of Knowledge Science, Japan Advanced Institute of Science and Technology, Nomi, Ishikawa 923-1211, Japan
[2]School of Manufacturing Systems and Mechanical Engineering, Sirindhorn International Institute of Technology, Thammasat University, Khlong Nueng, Pathum Thani 12120, Thailand

Corresponding authors: Van-Nam Huynh (huynh@jaist.ac.jp) and Warut Pannakkong (warut@siit.tu.ac.th)

ABSTRACT Rescheduling is essential in real-world production to adjust schedules when significant disturbances render existing ones non-optimal. Manufacturers are often required to reschedule production tasks as quickly as possible. This paper proposes a rapid production rescheduling framework for flow shop under machine failure disturbance, called PPGA-ANNs, with the goal of minimizing makespan while ensuring sufficient computational efficiency. The framework begins with a scheduling knowledge creation phase conducted before starting production. It applies the proposed Perturbation Population Genetic Algorithm (PPGA) to solve generated scenarios of flow shop production with machine failure problems. The performance of the PPGA is compared to other research algorithms and to the standard genetic algorithm (GA). The same data set from a widely used scheduling benchmark is used for all algorithms to confirm the effectiveness of the PPGA. Artificial neural networks (ANNs) are then applied to store the scheduling knowledge obtained from the PPGA. In the knowledge implementation phase, when a machine failure problem occurs during production, the rescheduling solution is provided by the ANNs if the machine failure problem is identical to a generated scenario. Otherwise, the rescheduling solution is provided by the PPGA, using the initial solution obtained from the ANNs. Based on the experimental results, the PPGA-ANNs framework demonstrates better performance in makespans than benchmark algorithms. Additionally, it provides faster solutions, particularly for new machine failure problems. In conclusion, the proposed framework is capable of minimizing the makespan with a short computational time for real-world production, addressing the limitations of existing state-of-the-art meta-heuristic algorithms.

INDEX TERMS Artificial neural network, flow shop production, genetic algorithm, machine failure, production rescheduling.

I. INTRODUCTION

Rescheduling problems of production systems are a form of decision support that plays a crucial role in actual dynamic manufacturing, where an unexpected situation can arise at any time. In the current competitive environment, effective production scheduling and rescheduling have become important for supporting innovative production methods and maximizing resource utilization for survival in the marketplace [1].

Predetermined schedules have been the norm in production and manufacturing for decades. However, achieving maximum productivity while adhering to the schedule has

The associate editor coordinating the review of this manuscript and approving it for publication was Wei Liu.

become challenging in today's manufacturing environment, which is fraught with uncertainties and complexities. Various theoretical studies have been conducted to analyze novel approaches to dynamic scheduling or rescheduling to mitigate the impact of intentional or unintentional disturbances on production processes. Nevertheless, practical applications in industry are rare [2].

In dynamic manufacturing, production rescheduling is typically required to moderate the consequences of disturbed events while maintaining an optimum performance [3]. Significant disturbances in production, such as machine failure, urgent job arrivals, and changes in due dates, make previously established schedules infeasible and hence are referred to as rescheduling factors [4]. These factors pose a significant challenge to manufacturers, who are required to perform rescheduling activities quickly and efficiently to maintain high-quality production. Among these factors, machine failure is a common occurrence in manufacturing, which can lead to suboptimal machine performance due to changes in the processing time. However, many researchers in production scheduling have assumed that machines are continuously available during the planning horizon, which is a significant deviation from actual production environments where unstable machines can delay product launch [5].

Among the initiatives of Industry 4.0, new perspectives and challenges in research on dynamic production scheduling have received increasing attention [6]. This research proposes a framework to address rescheduling problems in a flow shop production with machine failure by using a hybrid algorithm that combines a meta-heuristic algorithm, represented by the proposed genetic algorithm, and a supervised learning approach, represented by an artificial neural network (ANN). The proposed framework aims to generate a new schedule in rapid computational time with minimized makespan when a machine failure occurs. To achieve this objective, the present research first produces scenarios of machine failure in a large-scale flow shop production. Each scenario is solved, and an appropriate schedule is found using our proposed genetic algorithm, the Perturbation Population Genetic Algorithm (PPGA). The PPGA's performance is evaluated and compared with existing algorithms using the same data set from Taillard's benchmark [7], which generates scheduling problems of sizes that exceed the rare instances published. Second, the ANNs are trained using the knowledge gained from the machine failure scenarios and the suitable schedule for each scenario. Consequently, a predicted schedule can be obtained from the trained ANNs. In case the predicted schedule is infeasible, the PPGA is still applied by using the predicted schedule from the trained ANNs as an initial solution to achieve a feasible schedule in a more efficient computational time. The proposed framework is called PPGA-based knowledge obtained from the trained ANNs (PPGA-ANNs). Finally, the optimality and computational time of the proposed framework are compared to those of the standard genetic algorithm (GA) and PPGA without the initial solutions from the trained ANNs.

The remainder of this paper is organized as follows: Section II provides a comprehensive review of the relevant literature on production scheduling and rescheduling. Section III provides a detailed description of the flow shop production problem. Section IV proposes a framework to generate rapid production rescheduling, with the aim of mitigating the impact of machine failures on the performance of dynamic manufacturing systems. Section V presents the numerical implementation and experimental results to verify the effectiveness of the proposed framework. Section VI offers a comprehensive discussion and analysis of the findings and implications derived from the experimental results. Finally, Section VII presents the conclusions and suggests potential avenues for future research.

II. RELEVANT RESEARCH

Existing methodologies and techniques employed in the academic literature for production scheduling and rescheduling problems serve as valuable resources for identifying gaps and limitations in the current knowledge base. Moreover, they aid in recognizing potential areas for integrating production rescheduling systems. This literature review aims to present a comprehensive survey of previous research on the methodology of the problem domain, providing a clear understanding of the need for content deliberation, position issues, and the motivation behind the methodology developed in this research. First, we provide a concise overview of previous research on production scheduling, emphasizing the limitations of previous methodologies. Subsequently, we present new perspectives and existing research pertaining to the production rescheduling problem.

A. PRODUCTION SCHEDULING

Production scheduling has been the focus of research in various domains for several decades. It is defined as the problem of determining an optimal schedule or sequence for a set of jobs on a production line [8]. Moreover, based on a review of the literature, the production scheduling problem is widely recognized and classified as NP-hard [9].

The scheduling procedure also depends on the type of environment, such as a single machine, parallel machine, flow shop, or job shop [10]. Flow shop operation is one of the most common production environments, used in several industries such as automotive manufacturing [11], automobile manufacturing [12], plastic products [13], and wood industry [14]. Since the 1950s, many researchers have extensively explored algorithms for developing production sequencing problems to minimize makespan. In 1954, Johnson [15] introduced a simple algorithm for two-machine flow shop scheduling. Since then, scheduling has become an independent field of research with various algorithms. While exact optimization algorithms are widely presented for scheduling (e.g., [16], [17], [18]), the complexity of production scheduling problems renders

exact solvers infeasible of high-quality solutions for large-scale problems. In practice, optimal solutions are hardly required for moderate-and large-scale problems [19]. Heuristic algorithms are more feasible (e.g., [20], [21], [22]), but the computational time still takes remarkably long for even moderate-scale problems. Consequently, many researchers have turned to meta-heuristic algorithms that are feasible for searching near-optimal solutions and achieving them in a suitable amount of time.

Based on the literature, we can indicate famous meta-heuristic algorithms to address flow shop scheduling, such as tabu search [23], iterated greedy [24], simulated annealing [25], the ant colony optimization algorithm [26], GA [27], and recently using an algorithm inspired by a mathematical operator called the arithmetic optimization algorithm [28]. Among meta-heuristic algorithms, the GA is currently the most widely used and efficient for solving operational management problems, such as scheduling [29]. The GA is inspired by the biological evolution process and is commonly used to respond to optimization problems in large and continuous search spaces [30].

GA has gained significant popularity due to its ability to efficiently solve scheduling problems within reasonable computational time [31]. Previous studies have proposed various GA-based approaches to tackle different scheduling problems in diverse domains. Meena et al. [32] utilized GA to minimize workflow execution costs while meeting deadlines in a cloud computing environment. Peng et al. [33] introduced a hybrid GA to address the parcel delivery routing and scheduling problem. In the context of dynamic production, Yu et al. [27] proposed a GA incorporating a new decoding method specifically designed for the total tardiness objective, enabling the generation of tight schedules and solving the hybrid flow shop scheduling problem. Additionally, Wang and Zhu [34] employed a mathematical model, GA, and tabu search to address the flexible job shop environment, considering sequence-dependent set-up times and job lag times. These studies highlight the versatility and effectiveness of GA-based approaches in solving various scheduling challenges across different application domains.

Although the GA has shown great success in production scheduling problems, it is prone to being trapped in local optima or experiencing early convergence when exploring the search space [35], [36]. Accordingly, the proposed PPGA was developed to overcome this limitation. Moreover, although meta-heuristic algorithms excel at providing optimal solutions for large-scale production scheduling, they may not be able to satisfy the computational time requirements for rapid production rescheduling when unforeseen circumstances arise. In addition, most scheduling research fails to consider the various execution issues that may arise when implementing global manufacturing, assuming that the algorithm will execute the schedule exactly as it was created. Therefore, this research aims to enhance the GA as a meta-heuristic algorithm to achieve a suitable and rapid solution for rescheduling problems.

B. PRODUCTION RESCHEDULING

In dynamic production environments, belatedness can be caused by unpredictable disturbances in the manufacturing operations. The exigency for a new schedule should be examined to alleviate the impact of disturbances and fulfillment of customer commitments. The process of updating the production schedule under a flexible environment with shop floor disruptions is referred to as production rescheduling. These challenges pose significant obstacles and have given rise to a growing array of rescheduling factors. Existing scheduling approaches require considerable time to adjust the original schedules and reschedule unprocessed work orders in order to rapidly achieve a new optimal solution [19]. Consequently, the field of rescheduling problems continues to garner attention from researchers in academia and industry. The production rescheduling has been motivated by the strong application background and has been investigated by several reviewer authors such as Vieira, Herrmann, and Lin [3], Ouelhadj and Petrovic [37], Cardin et al. [38], Uhlmann and Frazzon [2], and Larsen and Pranzo [39].

Numerous studies have addressed the challenges of production rescheduling problems. Sabuncuoglu and Goren [40] introduced proactive scheduling, which integrates decision theory to understand robustness and stability measures in terms of when-to-schedule and how-to-schedule with rescheduling policies. The Wilkerson-Irwin algorithm and two heuristic algorithms based on an active schedule generation procedure were presented by Dong and Jang [41] to minimize mean tardiness of production job shop rescheduling caused by machine breakdowns, by enabling the change in the processing sequence of operations to utilize idle machine time. Kundakci and Kulak [42] developed a hybrid GA for a dynamic job shop scheduling problem with machine breakdowns, new order arrivals, and changes in processing time. The algorithm has been shown to deliver high-quality solutions within an efficient computational time.

The use of Artificial Intelligence (AI)-based approaches in various research fields has been highly successful and has opened up opportunities to explore alternative solutions for optimization problems. Deep learning has been utilized to design optimization algorithms by learning through the distribution of problem instances. Zhang et al. [43] proposed a fuzzy neural network that adapts a rescheduling decision model based on current system states and disturbances; however, this approach has not yet been implemented in a manufacturing system or tested in production. Nazari et al. [44] tackled the vehicle routing problem (VRP) using reinforcement learning to train the network by computing the rewards of outputs and incorporating an attention mechanism that addresses various parts of the input.

Along with Industry 4.0, Li et al. [45] proposed an integrated approach of machine learning (ML) classification and optimization algorithms for identifying rescheduling patterns in terms of when to reschedule and adopt GA to calculate the initial schedule for the flexible job shop scheduling problem. Recently, during the novel coronavirus pneumonia

(COVID-19) pandemic, Wu et al. [19] presented a neural network with reinforcement learning for scheduling the emergency production of medical masks. Although this proposed algorithm processes the schedule within a short time, its limitation is a baseline in reinforcement learning that need much room to be improved.

Our research recognizes the importance of further investigation in the field of production rescheduling. By emphasizing the need for additional research, our research contributes to the ongoing discourse and advocates for intensified research efforts to enhance the techniques, algorithms, and methodologies used in production rescheduling. This acknowledgment underscores the significance of continuous improvement and the pursuit of innovative solutions to optimize manufacturing scheduling practices.

The primary contribution of our research is the development of the PPGA and its integration with ANNs, forming a novel approach for rapid production rescheduling in flow shop environments under machine failure disturbance. The PPGA-ANNs framework offers distinct advantages over existing literature. The proposed PPGA aims to overcome the limitations of the GA, which avoids local optima and improves optimization scheduling performance. Additionally, the integration of ANNs aims to enhance rescheduling performance, particularly in terms of faster computational time.

Moreover, the rescheduling problem in flow shop production has not received extensive investigation in the literature. Given its complexity, precise meta-heuristics and deep learning techniques are necessary to propose solutions within a rapid computational time. Therefore, the development of an integrated rescheduling framework for flow shop production constitutes another valuable contribution of this research.

In conclusion, the proposed PPGA-ANNs framework offers several advantages, including enhanced optimization capabilities, knowledge utilization, and superior performance, distinguishing it from previous research and contributing to the advancement of the field of manufacturing scheduling.

III. PROBLEM DESCRIPTION

One of the most common production environments is the flow shop operation. The scheduling problem in a flow shop with m machines is recognized to be strongly NP-hard for $m \geq 3$ [46]. The flow shop production problem can be formulated as an $n \times m$, where n jobs must be scheduled for line production on m different machines. All machines (M_i) and jobs (J_j) are available at the start of production and commence as soon as possible for $i = \{1, 2, \ldots, m\}$ and $j = \{1, 2, \ldots, n\}$. Each job undergoes processing on all machines in the same sequence, starting at M_1, then proceeding to M_2, and so forth up to M_m. The processing time required for each job on a specific machine is denoted by p_{ij}, where (i, j) refers to the operation of job j on machine i. In this research, setup time is included in processing time. Additionally, it is important to note that each machine can only process one job at the time, and the buffer capacity between machines is assumed to be unlimited. The objective of the flow shop scheduling in this research is to minimize the completion time of the last job, also known as makespan, denoted by C, defined mathematically as:

$$min \quad C$$

The significant scale of flow shop production in rescheduling problems presents challenges in finding optimal solutions due to the inherent complexity and time-consuming nature. The extended processing time resulting from machine failure requires manufacturers to promptly adjust production plans to minimize makespan. However, these rescheduling problems often prove intractable for exact optimization algorithms and also pose computational challenges for heuristic-based search algorithms. In response to the practical need for rapid rescheduling in manufacturing, this research presents the proposed PPGA-ANNs framework designed to address the rescheduling problem in flow shop production. The framework aims to fulfill the requirements of efficient rescheduling and enhance overall production performance in dynamic manufacturing environments.

IV. METHODOLOGY

This section presents the detail of the proposed PPGA-ANNs framework for production rescheduling. The framework is divided into two main phases (i.e., knowledge creation and knowledge implementation), as illustrated in Fig. 1.

In the knowledge creation phase, before the production process begins, the framework focuses on training the ANNs to capture and store rescheduling knowledge. This is achieved by utilizing the rescheduling solutions generated by PPGA under various simulation scenarios of machine failures. Through this training process, the ANNs acquire the ability to comprehend and retain valuable insights and patterns, which can subsequently be employed to generate highly effective rescheduling solutions.

Once the production process begins and a machine failure occurs, the knowledge implementation phase is activated. In this phase, the trained ANNs are used to propose appropriate rescheduling solutions based on the specific machine failure scenario encountered. The trained ANNs are directly deployed to generate rescheduling solutions an actual machine failure scenario aligns with the simulation scenarios, denoted as *Case 1*.

In the event where the machine failure situation deviates from the simulation scenarios employed during the training process, denoted as *Case 2*, the solution generated by the ANNs is used as the initial solution for PPGA. By incorporating the knowledge acquired from the ANNs, PPGA is endowed with a more informed starting point for its optimization process. This integration of PPGA and ANNs facilitates a swift search and optimization process, enabling the identification of rescheduling solutions that are close to optimal within a reduced computational time.

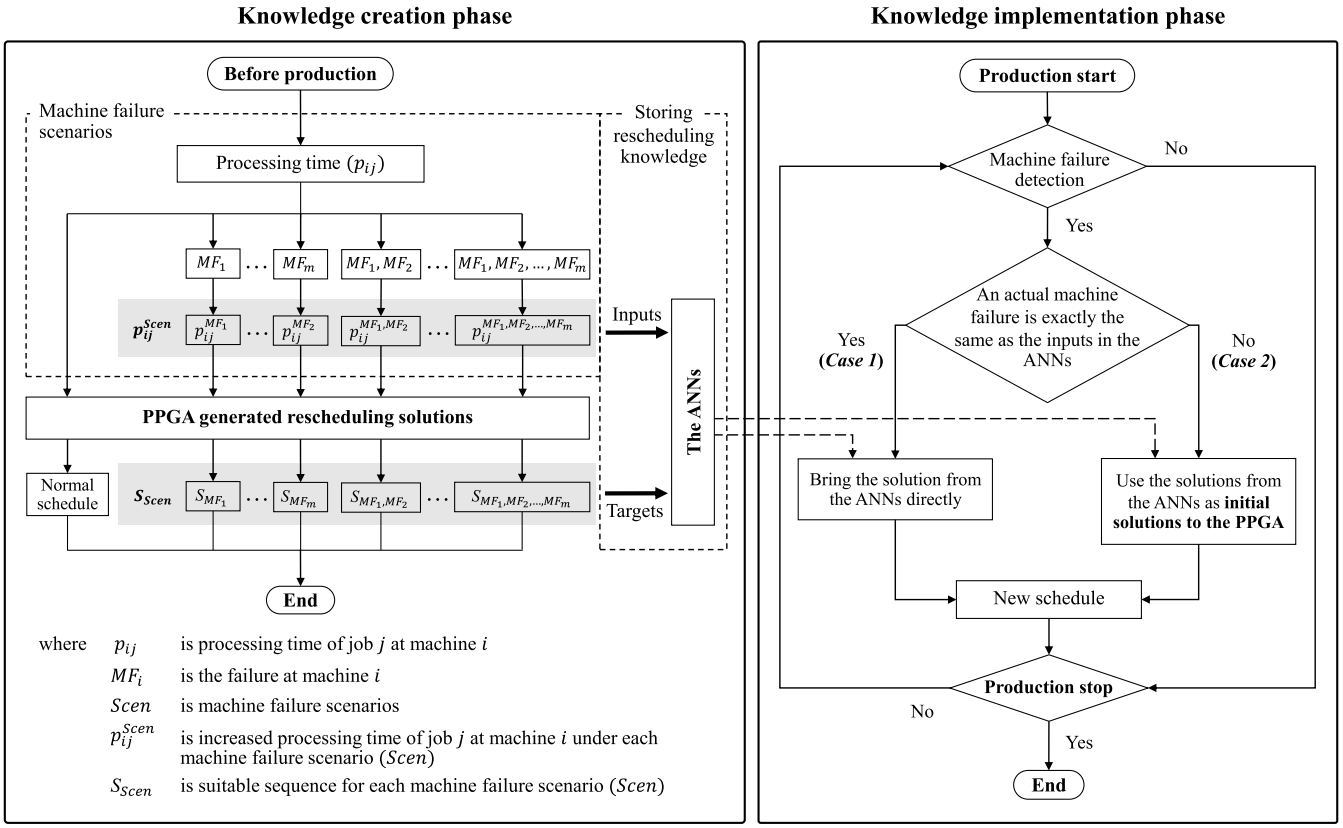


FIGURE 1. The proposed PPGA-ANNs framework for production rescheduling.

The detailed of the proposed PPGA-ANNs framework is provided in the subsequent subsections. Subsection IV-A presents the knowledge creation phase, encompassing the simulation of machine failure scenarios, the PPGA for generating rescheduling solutions, and the storing rescheduling knowledge by the ANNs. Subsection IV-B elucidates the knowledge implementation phase, encompassing the utilization of initial solutions resulted by the ANNs and the implementing of rescheduling knowledge from the ANNs into the PPGA.

A. KNOWLEDGE CREATION PHASE

The knowledge creation phase is conducted to create the knowledge from the rescheduling solutions produced by PPGA under various simulation scenarios of machine failure, and store the knowledge in the ANNs via the training process before starting the real production. The detail of the simulation scenarios is described in Section IV-A1. Section IV-A2 provides an explanation of the PPGA used to generate the rescheduling solutions. Finally, the ANNs utilized for storing the acquired knowledge are elucidated in Section IV-A3.

1) MACHINE FAILURE SCENARIOS

The disturbance event addressed in this research is the impact of machine failure on manufacturing processing time. It is assumed that each machine has a risk of failure. The processing time of each job j on machine i under the normal production is denoted by p_{ij}, where $i = \{1, 2, \ldots, m\}$ and $j = \{1, 2, \ldots, n\}$. The p_{ij} is defined as an $n \times m$ matrix.

In the scenario of machine failure, we assume that processing times will be increased. The increased processing time is represented by p_{ij}^{Scen}, and that machine overhaul is not required. $Scen$ represents the set of all possible scenarios of machine failure. A member of $Scen$ is a scenario of machine failure consisting of a set containing the occurrence of all failed machines. The occurrence of machine i is denoted by MF_i. In each scenario of machine failure, it is possible to have a single or multiple failed machines. The members of $Scen$ are generated as the power set of i, excluding the empty set, denoted by $\mathcal{P}\{i\} - \emptyset$. Therefore, the total number of all possible scenarios of machine failure is $2^m - 1$, where m denotes the total number of machines. The member of $Scen$ can illustrate as $\{\{MF_1\},\ldots,\{MF_m\}, \{MF_1, MF_2\},\ldots, \{MF_1, MF_m\},\ldots, \{MF_1, MF_2, \ldots, MF_m\}\}$. In addition, the impact of machine failure is quantified as q_i, which represents the tolerance level for increased processing times. The tolerance level varies depending on the acceptance criteria of individual manufacturers (Q) and does not require production to be stopped. The acceptance of the tolerance level is represented by the positive real number ($Q \in \mathbb{R}^+$). Therefore, the increased processing time (p_{ij}^{Scen}) is calculated using (1).

$$p_{ij}^{Scen} = q_i \times p_{ij} \quad ; \quad 1 < q_i \leq Q \tag{1}$$

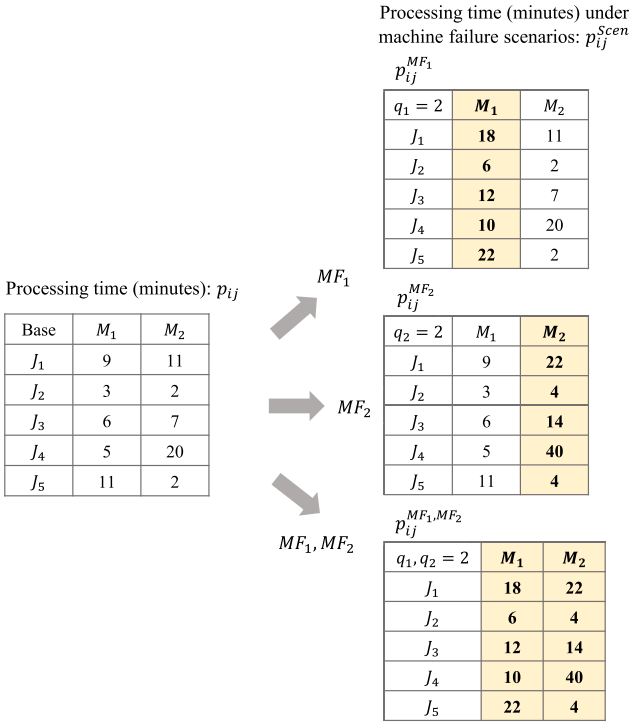


FIGURE 2. Example of machine failure scenarios.

For example, consider the machine failure scenarios presented in Fig. 2, where there are five jobs denoted as $j = \{1, 2, \ldots, 5\}$, and two machines denoted as $i = \{1, 2\}$. In this case, the set of possible machine failure scenarios includes $\{MF_1\}$, $\{MF_2\}$, and $\{MF_1, MF_2\}$, which are generated using the power set of machines excluding the empty set, resulting in $2^2 - 1 = 3$ scenarios. The impact of machine failure is quantified as q_i, and for this example, it is assumed to have a value of two. This means that in the event of failure, the processing time (p_{ij}^{Scen}) will be twice the normal processing time (p_{ij}). Consequently, if machine 1 fails (MF_1), it will result in the processing time ($p_{1j}^{MF_1}$) being doubled, while the other machines continue to operate at their normal processing times.

2) PPGA GENERATED RESCHEDULING SOLUTIONS

This section presents the proposed genetic algorithm named the Perturbation Population Genetic Algorithm (PPGA). The PPGA is built upon the standard genetic algorithm (GA) to enhance its capabilities.

GA derives inspiration from the biological evolution process and has demonstrated remarkable effectiveness in tackling problems with vast search spaces, such as scheduling problems in manufacturing. It is widely used among meta-heuristic algorithms for addressing NP-hard problems and has been shown to produce outstanding and effective results. Additionally, the speed of the algorithm makes it suitable for use in real-world manufacturing decisions. However, it is noteworthy that although the GA demonstrates proficiency in exploring solutions within complex spaces through global search, it is also susceptible to becoming entrapped in local optima during exploiting local search [35], [36].

Algorithm 1 Pseudocode of the PPGA

Require: Processing time (p_{ij}), Population size (P), Crossover probability (Cr), Mutation probability (Mu), Number of iterations (Itr), Maximum runtime ($Mrun$), Percentage of improvement (γ), Number of compared iterations (β), Number of perturbations (Ptb)
Ensure: Position of jobs in sequence (S)

```
Iteration = 0
Perturbation = 0
x = 0
while Runtime ≤ Mrun do
  while Iteration < Itr do
    if x < β then
      Population initialization
      Crossover operator
      Mutation operator
      Fitness value calculation
      Selection operator
      Iteration = Iteration + 1
      if fitness value improves ≤ γ% then
        x = x + 1
      else
        x = 0
      end if
    else if Perturbation < Ptb then
      Perturb population initialization
      Perturbation = Perturbation + 1
      x = 0
    else
      x = 0
    end if
  end while
end while
```

In this research, we developed the PPGA to address the rescheduling problem of machine failure in flow shop production. In the knowledge creation phase, the PPGA is employed to obtain the rescheduling results under various scenarios of machine failure. The overview of the PPGA is depicted in Fig. 3, which provides a visual representation of its functioning and components. Moreover, the pseudocode of the PPGA algorithm is presented in Algorithm 1, outlining the step-by-step procedure and operations involved in its execution. Initially, the rescheduling results are generated in a standard genetic operation stage. To avoid the issue of local optima entrapment during the schedule search process, this research proposes a perturbation operation stage incorporating with the standard genetic operation stage.

a: STANDARD GENETIC OPERATION STAGE

The standard genetic operation stage is the first stage of the PPGA. It utilizes the standard genetic operators such as

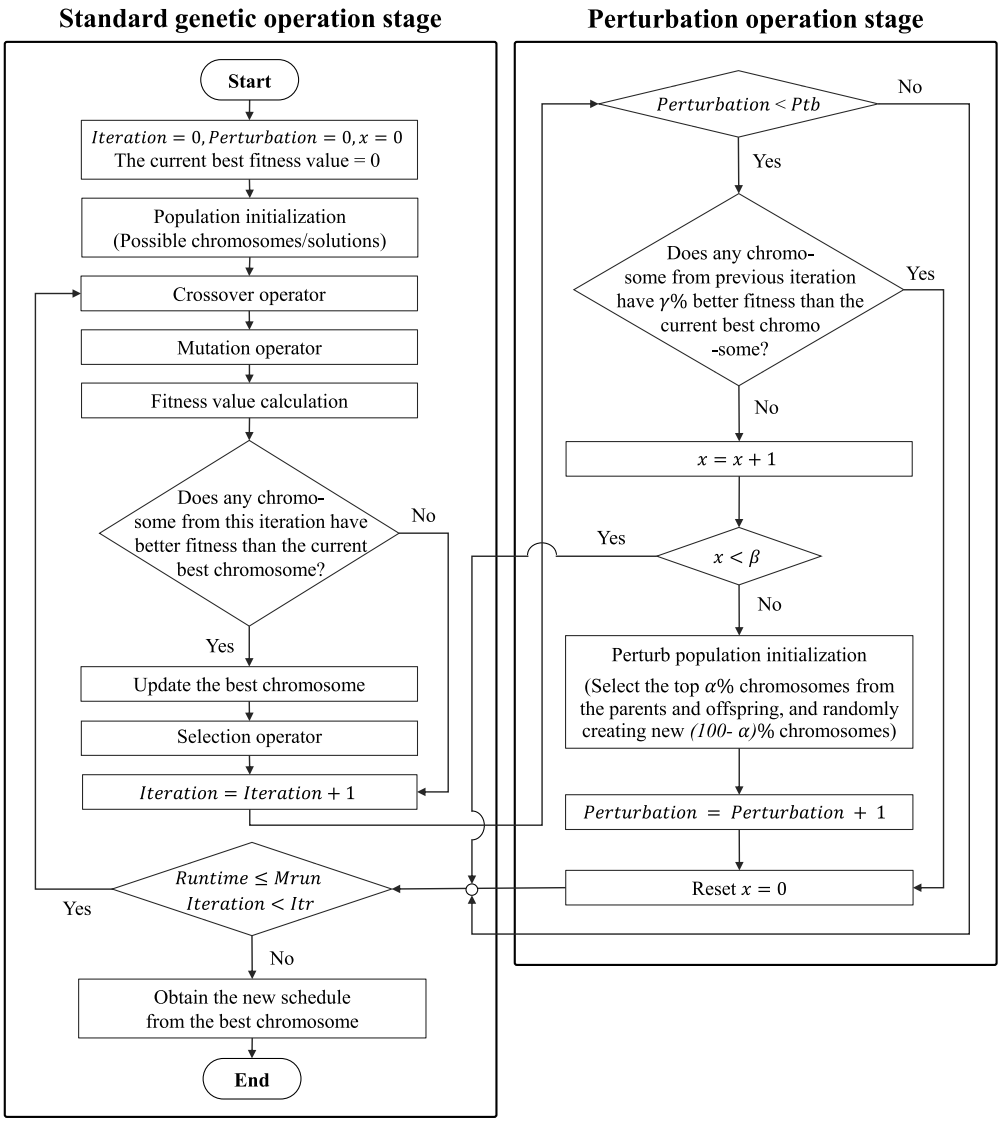


FIGURE 3. The proposed PPGA workflow.

population initialization, crossover operator, mutation operator, fitness value calculation, and selection operator, to generate the rescheduling results. The details of this stages are explained in the following paragraphs.

Chromosome representation: Traditionally, the chromosome representation of the GA, consisting of binary genes, wherein each gene can assume a value of either 0 or 1, is unsuitable for scheduling problems [47]. In this research, the chromosome representation for the population of possible solutions in the PPGA is redesigned as a sequence of jobs for flow shop production. A set of individual chromosomes is referred to as a population, which represent possible solutions to the scheduling problem. Each chromosome is characterized by a set of parameters known as genes.

Fig. 4 illustrates a representation of a population consisting of four chromosomes. Suppose there is a chromosome fragment (i.e., gene) of "5 3 2 4 1," indicating that the first

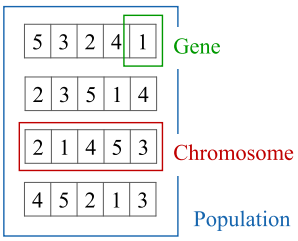


FIGURE 4. Chromosome representation.

position in this sequence is job number "5", the second position is job number "3", followed by job number "2", "4", and "1," respectively.

Population Initialization: The population initialization is the first step in the standard genetic operation stage. The initial population (a subset of all solutions) is generated in

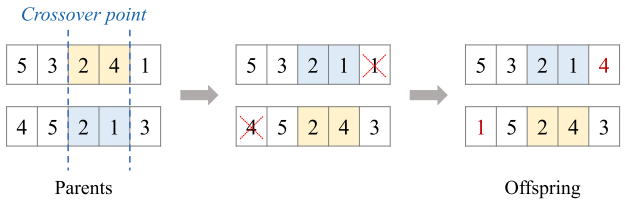


FIGURE 5. Two-point crossover operator.

this step. The population can be defined as a set of chromosomes. As reported by Konak et al. [48], diversity within the population can lead to an approximate global optimal solution. The population must be diverse in order to prevent premature convergence, where the algorithm converges before the optimal solution is reached. In order to ensure the significance of diversity, random initialization is the one to drive the population towards optimality [49]. Hence, the population is initialized with entirely random solutions.

Previous research has recommended that the suitable population size (P) should be determined through experimentation. It is important to note that maintaining a very large population size may slow down the algorithm, whereas a smaller population size may not be sufficient for a mating pool. Therefore, determining an optimal population size requires trial and error.

Crossover Operator: The crossover operator, also known as recombination, is an essential operator of the GA. To determine whether to apply the crossover operator on a given chromosome pair, denoted by $pair = \{1, 2, \ldots, P/2\}$, the algorithm evaluates a random probability for each chromosome pair, denoted by $ProbC_{pair}$, against a predetermined crossover probability, Cr. If $ProbC_{pair}$ is less than Cr, the algorithm executes the crossover operator; otherwise, the crossover operator is not applied for that particular pair. It should be noted that the optimal value of Cr depends on the specific problem and population characteristics. Typically, in the hyperparameter tuning process, the value of Cr ranges between 0.5 and 1.0 [50].

The crossover operator involves merging genes from two chromosomes, referred to as parents, to produce two novel chromosomes, known as offspring. In this research, we employ a two-point crossover operator, as shown in Fig. 5.

As the chromosomes are encoded as actual numbers (job numbers), exchanging parts of a chromosome may result in the occurrence of duplicate gene fragments in the offspring chromosomes (see job 1 and 4 in Fig. 5). Because our problem is related to flow shop production, where each task position should be located only once, such duplication is not allowed. To address this issue, as depicted in the example, the duplicated genes are randomly replaced by leftover genes after the selected genes are swapped. Therefore, the crossover operation is deemed complete once the offspring are generated.

Mutation operator: The mutation operator in the GA is intended to mitigate the risk of the algorithm becoming trapped in local optima by promoting genetic diversity from one generation of chromosomes to the subsequent generation. Fig. 6 illustrates the mutation applied to the PPGA.

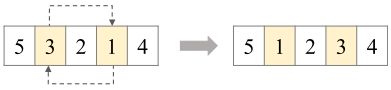


FIGURE 6. Swapping mutation operator.

In the PPGA, the decision to perform the mutation operation on a given chromosome, denoted by $c = \{1, 2, \ldots, P\}$, is determined by comparing a random probability, denoted by $ProbM_c$, with the mutation probability, denoted by Mu. If $ProbM_c$ is less than Mu, the algorithm executes the mutation operator on that specific chromosome; otherwise, the mutation operator is not applied. However, because the mutation operator can potentially lead to a loss of diversity and negatively impact performance, it is typically assigned a lower probability compared to the crossover operator. The value of Mu ranges from 0.001 to 0.05 [50]. After a chromosome is selected for mutation, swapping mutation is used as the mutation operator. This involves exchanging genes in the chromosome by randomly selecting a pair of different positions.

It is important to note that although the mutation operator can sometimes lead to substantial changes in the chromosome structure, it is not guaranteed that the fitness value of the individual chromosome will improve after undergoing mutation. Nevertheless, the mutation operator is valuable for enhancing the overall optimization control of the algorithm [51].

Fitness Value Calculation: The fitness function plays an essential role in evaluating the quality of chromosomes in the PPGA. The fitness value has a direct impact on the probability of individuals being chosen to complete genetic operations [52]. A well-designed fitness function can accelerate convergence and increase the likelihood of reaching the optimal solution.

In the context of flow shop scheduling, the fitness value of chromosome c, denoted as f_c, is calculated using the makespan of chromosome c, denoted as C_c, as follows.

$$f_c = \frac{1}{C_c} \tag{2}$$

The objective of the PPGA is to determine a chromosome c that maximize the fitness value f_c by minimizing the makespan C_c, which is equivalent to minimizing the total idle time on the last machine (machine m), in the flow production line [10]. The makespan of chromosome c can be computed as follows:

$$C = \sum_{i=1}^{m-1} p_{i1} + \sum_{k=1}^{n-1} I_{mk} + \sum_{j=1}^{n} p_{mj}, \tag{3}$$

which is the summation of three terms. The first term is the idle time of the last machine before starting the first job in the sequence where p_{ik} denotes the processing time on machine i of the job in the k^{th} position in the sequence by $k = \{1, 2, \ldots, n\}$. The second term is the idle time of the last

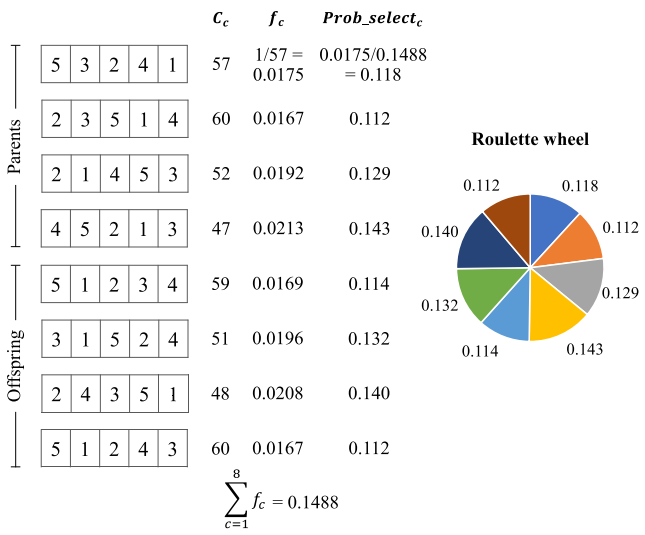


FIGURE 7. Roulette wheel selection operator.

machine between the jobs in the sequence where I_{ik} indicates the idle time on machine i between the operation of the jobs in the k^{th} position and $(k+1)^{\text{th}}$ position. The last term is the total processing time of the last machine where p_{ij} is processing time of each job j on machine i. However, since the processing time at the last machine (the last term) remains constant regardless of the job sequence, the algorithm can focus on optimizing only the first two terms in order to minimizing the overall value.

Selection Operator: The selection of chromosomes for recombination is used to determine which individuals will be selected for breeding in the next iteration. Our PPGA uses roulette wheel selection, which is a widely used selection method in GA [53].

Each chromosome is assigned a section on a roulette wheel in proportion to its corresponding fitness value (see Fig. 7). The roulette wheel selection favors chromosomes with higher fitness values, increasing the chances of being selected for reproduction. Therefore, the larger the fitness value, the larger the section occupied by the corresponding chromosome on the wheel.

After assigning sections to each chromosome, the wheel is spun, and the chromosome in the section where the pointer lands is selected for reproduction in the next iteration. This process is repeated until the desired number of chromosomes, denoted by population size P, is reached. The probability of selecting an individual chromosome ($Prob_s elect_c$) is calculated using the following formula:

$$Prob_s elect_c = f_c / \sum_{c=1}^{A} f_c, \tag{4}$$

where A denotes the number of parents and offspring, and f_c denotes the fitness value of the individual chromosome c. Fig. 7 illustrates an example of the roulette wheel selection process, where both the parent and offspring populations comprise four chromosomes. The chromosome with a higher fitness value has a higher proportion on the roulette wheel, thereby increasing its chance of being selected for reproduction.

The roulette wheel selection is an effective and simple method for maintaining genetic diversity while favoring individuals with higher fitness values. However, this may lead to the elimination of all offspring from the crossover operator. In this case, only the parents are passed through the next iteration, resulting in an increased likelihood of converging to a local minimum. This lack of diversity poses a challenge for identifying globally optimal solutions. Therefore, it is necessary to introduce perturbations to the population to maintain genetic diversity and improve the chances of finding globally optimal solutions.

b: PERTURBATION OPERATION STAGE

The perturbation operation stage is a component of the PPGA that aims to improve the algorithm's ability to escape from local optima and explore uncharted search spaces. This research proposes the perturbation operation stage that is triggered when the fitness value of the best chromosome in the current iteration does not improve by more than $\gamma\%$ compared with the best fitness value in the previous β iterations. This indicates that the algorithm has not made significant progress in the current iteration, potentially indicating a state of stagnation within the local optimum.

Once the perturbation operation stage is triggered, the top $\alpha\%$ of the chromosomes in the population based on fitness values are selected to preserve the best chromosomes from the population in the standard genetic operation stage. The remaining $(100 - \alpha)\%$ are removed from the population. After that, the new chromosomes are randomly generated to replace the removed population to enhance the population diversity and explore uncharted search spaces. Moreover, setting $(100-\alpha)$ to a value greater than α is possible to promote greater genetic diversity. Suppose the improvement of the chromosomes in the last β previous iterations is still lower than $\gamma\%$ in comparison to the best chromosome obtained from the standard genetic operation stage, the PPGA continues to perform the perturbation operation until the determined number of perturbations (Ptb) is reached.

By employing this population diversity strategy, the chance of obtaining an approximate optimal solution is potentially improved by increasing the chance of escaping a local optimum. Consequently, the standard genetic operators (i.e., crossover operator, mutation operator, fitness value calculation, and selection operator) are performed on the new population generated through the perturbation operation stage to obtain improved fitness values.

c: TERMINATION CRITERIA

The termination criteria refer to the condition or set of conditions that determines when the algorithm should stop searching for better solutions. Several termination criteria can be used in a GA, and the choice of criteria depends on the problem being solved and the available computational

4	5	2	1	3

$C = 47$

FIGURE 8. Solution from the PPGA.

resources. The termination criteria used should strike a balance between the quality of solutions and computational time. Choosing appropriate termination criteria is crucial to ensure that the algorithm produces good solutions within a reasonable computational time.

Using the maximum number of iterations as a termination criterion is a common approach for a GA. This criterion specifies a fixed number of iterations that the algorithm should be run before terminating. In addition, the maximum runtime can be used to terminate the algorithm within a fixed time frame. These two termination criteria are useful when there are time constraints or when the algorithm is run with limited computational resources.

In this research, the termination criteria of the PPGA are both the maximum number of iterations (*Itr*) and the maximum runtime (*Mrun*). It is important to choose a reasonable maximum number of iterations based on the complexity of the problem and efficiency of the PPGA. Therefore, the maximum number of iterations (*Itr*) is defined through the experiment. The maximum runtime (*Mrun*) depends on the time constraint provided by the user.

d: SOLUTIONS FROM THE PPGA

The primary objective of the PPGA is to locate the optimum of a given rescheduling problem by exploring different regions of the solution space. The PPGA expands the perturbation operation stage, enabling the algorithm to effectively navigate the solution space and mitigate the issue of convergence to a local optimum.

In the knowledge creation phase (Section IV-A), the PPGA is utilized to generate rescheduling solutions under various machine failure scenarios. The solution of the PPGA is an appropriate job sequence for each scenario, denoted by S_{Scen} where *Scen* is the set of all possible scenarios of machine failure, and evaluates their makespan as illustrated in Fig. 8. The solutions are then stored for further use by the proposed ANNs.

3) STORING RESCHEDULING KNOWLEDGE BY ANNs

The relevant research publications mentioned that the computational time of meta-heuristic algorithms could be more practical for rapid rescheduling in real-world manufacturing processes by incorporating with machine learning techniques. In this research, the incorporation between the PPGA and the ANNs is proposed to conquer the limitation of previous publications in the literature. The proposed ANNs are employed to store knowledge from the solutions generated by the PPGA.

The overall workflow of storing rescheduling knowledge by the proposed ANNs is illustrated in Fig. 9. The ANNs architecture is designed to extract the relationship between the given rescheduling problems (inputs) and the solutions

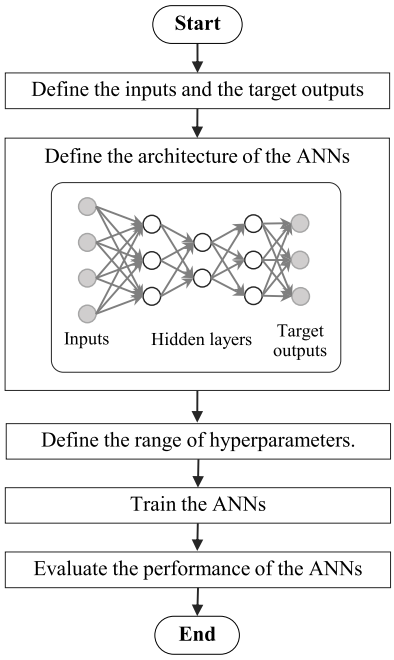


FIGURE 9. The ANNs workflow.

generated by the PPGA (target outputs). Each ANN is constructed to predict the suitable sequence for a job in a given rescheduling problem.Therefore, the total number of ANNs depends on the total number of jobs in the problem. The steps involved in storing rescheduling knowledge in the ANNs are as follows:

Step 1. Define the Inputs: In the ANNs, input layer is the initial layer and responsible for passing the input data to the subsequent layers. Each input node in the input layer represents an input attribute, and the values of these nodes correspond to the values of the input attributes. The input attributes are the processing times of each job in each machine. The number of input nodes is equal to the number of the input attributes. In this research, the input layer comprises $n \times m$ input nodes, where n denotes the number of jobs and m denotes the number of machines. The number of instances in the input data is the number of all possible machine failure scenarios.

All values of the input attributes are normalized using a standardization of data scaling method to avoid bias from different data scales among the input attributes. The method converts the values of each attribute into a standard normal distribution. In other words, the method subtracts each value by the mean of its corresponding attribute, and then divides the result by its standard deviation. The normalized values for each input is calculated through the following equation:

$$y_{ij} = \frac{p_{ij} - \overline{p_{ij}}}{\sigma_{ij}}, \tag{5}$$

$$\sigma_{ij} = \sqrt{\frac{\sum (p_{ij} - \overline{p_{ij}})^2}{N}}, \tag{6}$$

TABLE 1. Example of normalized inputs in the ANNs.

Scen	p_{ij}									
	J_1M_1	J_1M_2	J_2M_1	J_2M_2	J_3M_1	J_3M_2	J_4M_1	J_4M_2	J_5M_1	J_5M_2
Base	9	11	3	2	6	7	5	20	11	2
MF_1	18	11	6	2	12	7	10	20	22	2
MF_2	9	22	3	4	6	14	5	40	11	4
MF_1, MF_2	18	22	6	4	12	14	10	40	22	4
Mean ($\overline{p_{ij}}$) and standard deviation (σ_{ij}) calculation										
$\overline{p_{ij}}$	13.5	16.5	4.5	3	9	10.5	7.5	30	16.5	3
σ_{ij}	5.196152	6.350853	1.732051	1.154701	3.464102	4.041452	2.886751	11.54701	6.350853	1.154701
Normalization of the input value										
Scen	y_{ij}									
	J_1M_1	J_1M_2	J_2M_1	J_2M_2	J_3M_1	J_3M_2	J_4M_1	J_4M_2	J_5M_1	J_5M_2
Base	-0.86603	-0.86603	-0.86603	-0.86603	-0.86603	-0.86603	-0.86603	-0.86603	-0.86603	-0.86603
MF_1	0.866025	-0.86603	0.866025	-0.86603	0.866025	-0.86603	0.866025	-0.86603	0.866025	-0.86603
MF_2	-0.86603	0.866025	-0.86603	0.866025	-0.86603	0.866025	-0.86603	0.866025	-0.86603	0.866025
MF_1, MF_2	0.866025	0.866025	0.866025	0.866025	0.866025	0.866025	0.866025	0.866025	0.866025	0.866025

where y_{ij} represents the normalized value, p_{ij} represents the original input value of processing time, $\overline{p_{ij}}$ and σ_{ij} represent the mean and the standard deviation of the processing time, respectively, and N represents the number of machine failure's scenarios, $i = \{1, 2, \ldots, m\}$ and $j = \{1, 2, \ldots, n\}$ represent indexes of jobs and machines, respectively. Table. 1 illustrates an example of the normalization of the input values in the proposed ANNs.

Step 2. Define the Target Outputs: The output layer of the ANNs serves as the final layer responsible for producing predicted outputs. Ideally, once the values of the input nodes are processed through the ANNs, the values of predicted outputs should match the target outputs, which are solutions generated by the PPGA. These solutions are represented as sequences of jobs (Fig. 10 (a)); however, because of the ANN's ability to handle only numerical data, each solution (S_{Scen}) is converted into binaries. Each binary represents whether the job is assigned to a position in the sequence. The number of binaries generated from each solution depends on the number of positions that is equal to the number of jobs. In this research, the ANNs are constructed dedicatedly for each job, with the number of output nodes in each ANN being equal to the number of binaries.

To demonstrate the results from the binarization, the binary matrixes with dimension of $n \times n$ are created to represent target outputs (Fig. 10 (b)) from the sequences of the solutions (S_{Scen}) in the binary format, where n denotes the number of jobs. The elements of the matrixes, s_{jk}, denote whether job j is assigned to position k in the sequence where $j, k = \{1, 2, \ldots, n\}$. If job j is assigned to position k, then s_{jk} is set to 1; otherwise, it is set to 0. For example, in a scenario of five jobs ($j = \{1, 2, \ldots, 5\}$), five positions ($k = \{1, 2, \ldots, 5\}$), and two machines ($i = \{1, 2\}$), for each job, its positions in the four solutions (sequences) generated by the PPGA under all possible scenarios (i.e., base case, S_{MF_1}, S_{MF_2}, and S_{MF_1,MF_2}) are presented in a 5×5 matrix representing s_{jk} for five jobs and five positions (i.e., $s_{11}, s_{12}, \ldots, s_{15}, s_{21}, \ldots, s_{55}$). The arrows in Fig. 10 illustrate the connections between the original solutions from PPGA and the binarized solutions (target outputs) for job 1.

Step 3. Define the Architecture of the ANNs: The architecture of the ANNs includes input, hidden, and output layers, with the details of the input and output layers described in Steps 1 and 2. The number of the hidden layer can be a positive integer. An example of architecture of the ANNs for production of five jobs ($n = 5$) and two machines ($m = 2$) is presented in Fig. 11.

The input layer of the ANNs in this example consists of 10 nodes of the processing times from 5 jobs $\times$ 2 machines. Totally, five ANNs are constructed. The result from output nodes of each ANN is used for predicting the position of each job. This example consists of 5 output nodes representing the number of possible positions for that job. The number of nodes in the hidden layers is determined through fine-tuning experiments.

Step 4. Define the Range of Hyperparameters: Hyperparameters in an ANN are set before training and affect the behavior of the model. These parameters are not learned during training but are chosen based on domain knowledge, experimentation, or best practices.

1) Hidden layers and nodes: Hidden layers are intermediate layers between the input and output layers. They perform computations on the input data to extract relevant features and pass the results to the next layer. The number of hidden layers and nodes in each hidden layer are hyperparameters.
2) Activation function: The activation function is a mathematical function that transforms an aggregated input into an output for the nodes in the hidden layers and the output layer. It is possible to have multiple activation functions in the ANN. This allows the ANN to learn complex knowledge in the training data.
3) Learning Rate: The learning rate determines the step size at which the weights and biases of the ANN are updated during training. A higher learning rate allows for faster convergence but may result in overshooting

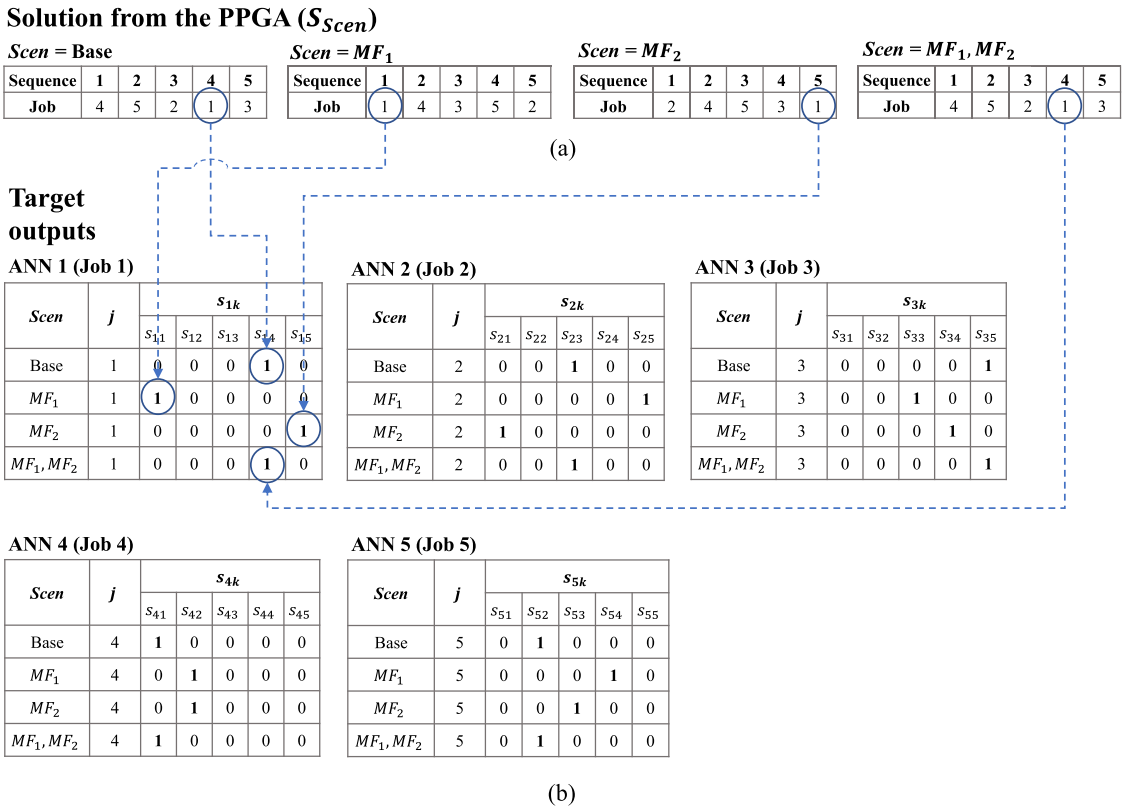


FIGURE 10. Example of the outputs in the ANNs. (a) Solutions from the PPGA and (b) Target outputs of all ANNs.

the optimal weights, while a lower learning rate may take longer to converge but may result in more accurate weights.

4) Epochs: The number of epochs determines how many times the training data is used to update the weights and biases of the ANN. Too few epochs may result in underfitting, where the ANN does not learn the underlying patterns in the data, while too many epochs may result in overfitting, where the ANN memorizes the training data but does not generalize well to new data. The suitable number of epochs depends on the problem's complexity and the data set's size.
5) Batch Size: In each epoch, the training data is divided into batches, and the weights and biases are updated based on the average error calculated over each batch. The batch size determines how many samples are used in each update step. A larger batch size may result in faster convergence but requires more memory, while a smaller one may result in slower convergence but requires less memory. The batch size depends on the available computational resources and the characteristics of the data set.
6) Optimizer: The optimizer is an algorithm used to update the weights and biases of the ANN during the training process in order to minimize the error or loss between the predicted and target outputs.

The values of the hyperparameters of the proposed ANNs are adjusted to minimize the loss between the predicted outputs and the target outputs. This results in an optimized model that can store the knowledge from PPGA and make accurate predictions on new rescheduling problems.

Based on the same architecture used by all ANNs for a specific number of jobs and machines, it is possible to either use the same values for the hyperparameters that are found to be optimal for a specific ANN (such as the ANN for job 1), or to vary the values by deviating from the optimal values of the ANN for job 1. The detail of the hyperparameter optimization process is shown in Fig. 12.

Step 5. Train the ANNs: The ANNs are trained using the machine failure scenarios that consist of inputs representing the processing times of jobs at specific machines, and target outputs representing the corresponding optimal production sequences resulting from PPGA. The entire data set of solution from PPGA is passed through the ANNs without separating it into training, validation, and test sets. This decision is based on the primary objective of the ANNs, which is to capture the knowledge inherent in the data set generated by PPGA, rather than to make predictions for new or unknown instances. Furthermore, this decision is due to the wide distribution of simulation scenarios, which encompass a diverse range of potential machine failure occurrences and their corresponding effects on the processing time.

Inputs for all ANNs

Scen	p_{ij}									
	J_1M_1	J_1M_2	J_2M_1	J_2M_2	J_3M_1	J_3M_2	J_4M_1	J_4M_2	J_5M_1	J_5M_2
Base	9	11	3	2	6	7	5	20	11	2

(a)

Target outputs

ANN 1 (Job 1)

Scen	*j*	s_{1k}				
		s_{11}	s_{12}	s_{13}	s_{14}	s_{15}
Base	1	0	0	0	**1**	0

ANN 4 (Job 4)

Scen	*j*	s_{4k}				
		s_{41}	s_{42}	s_{43}	s_{44}	s_{45}
Base	4	**1**	0	0	0	0

ANN 2 (Job 2)

Scen	*j*	s_{2k}				
		s_{21}	s_{22}	s_{23}	s_{24}	s_{25}
Base	2	0	0	**1**	0	0

ANN 5 (Job 5)

Scen	*j*	s_{5k}				
		s_{51}	s_{52}	s_{53}	s_{54}	s_{55}
Base	5	0	**1**	0	0	0

ANN 3 (Job 3)

Scen	*j*	s_{3k}				
		s_{31}	s_{32}	s_{33}	s_{34}	s_{35}
Base	3	0	0	0	0	**1**

(b)

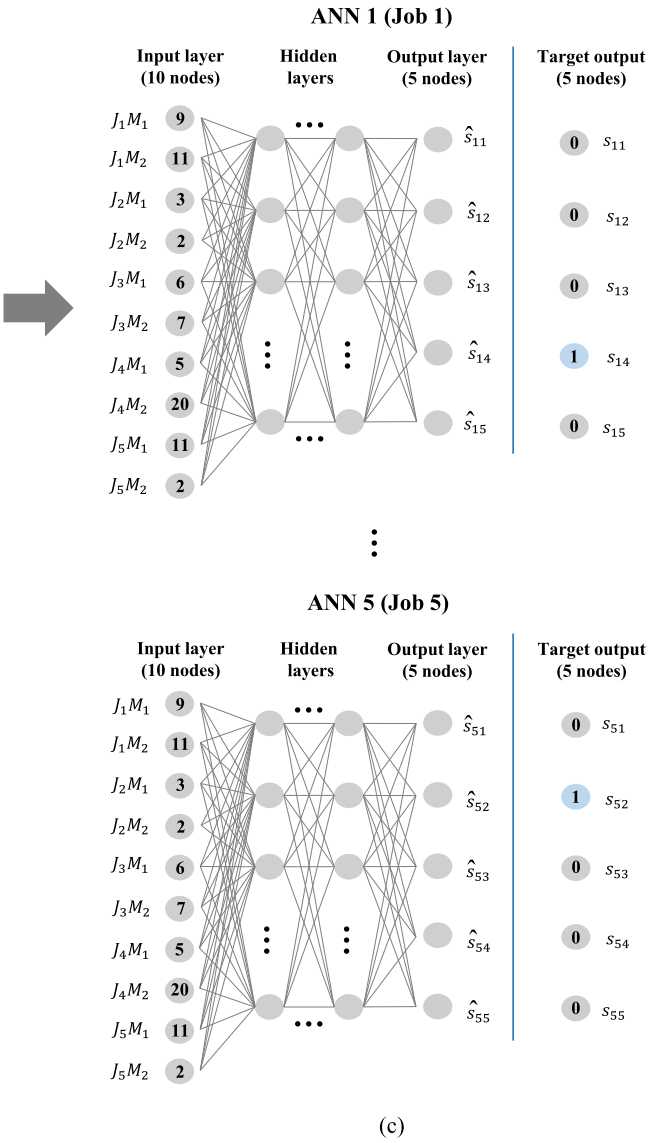


FIGURE 11. Example of the architecture of the ANNs. (a) Inputs for all ANNs, (b) Target outputs, and (c) The architecture of the ANNs.

During training, the ANNs perform forward propagation to calculate the predicted outputs of the nodes in each layer, starting from the input layer and propagating through the hidden layers to the output layer. This involves applying the activation function to the weighted sum of inputs for each node in each layer.

The ANNs subsequently update the weights and biases of the model to minimize a loss function, which serves as a performance measure during the training process. The loss function is essential in guiding the optimization process by identifying the quantity that the ANNs should minimize to accurately predict the outputs for the given inputs.

This research employs the *BinaryCrossentropy* loss function for optimization during the training process. This function measures the probability of dissimilarity between the predicted and target outputs and heavily penalizes the model for assigning high probability to the wrong class. This mechanism guides the ANNs toward making correct predictions and improving accuracy in binary classification problems, such as the one investigated in this research.

Step 6. Evaluate the Performance of the ANNs: The *Accuracy* metric is used to evaluate the performance of the trained ANNs. This metric provides a straightforward way to assess the ANN's prediction capability by comparing its predicted outputs with the target outputs of training data to obtain the number of correctly classified samples. Then, the *Accuracy* metric is calculated as the ratio of the number of correctly classified samples to the total number of samples in the training set. Based on the value of this metric, the performance of the ANNs can be evaluated in terms of overall correct classification.

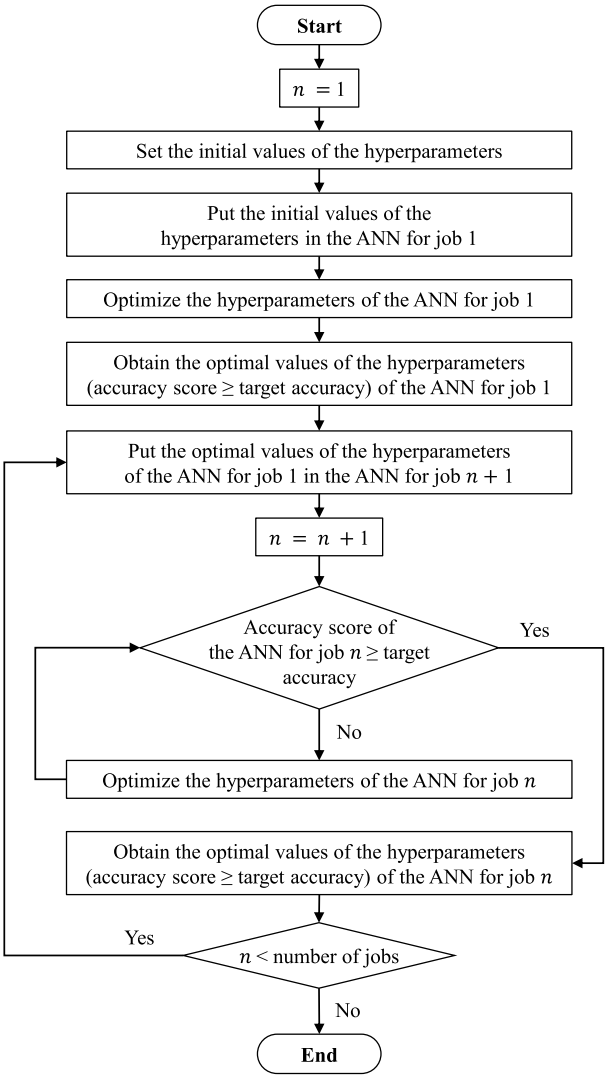


FIGURE 12. The optimization process of the hyperparameters in the ANNs.

B. KNOWLEDGE IMPLEMENTATION PHASE

The knowledge implementation phase is an important part of the proposed PPGA-ANNs framework (Fig. 1) as it involves utilizing the knowledge obtained during the knowledge creation phase to make decisions about rescheduling and minimizing the impact of machine failures on the production process. This phase is initiated upon the machine failure is detected during the production process. The machine failure can be classified as *Case 1* and *Case 2*. In *Case 1*, where the machine failure during production is identical to the inputs used in the trained ANNs, the new schedule can be directly obtained from the solutions provided by the trained ANNs.

Moreover, in *Case 2*, where the machine failure situation is different from the inputs used to train the ANNs (i.e., a new instance), the new schedule is obtained from PPGA by using the solutions from the trained ANNs as initial solutions in the PPGA's initial population. The process of predicting the solutions from the trained ANNs is presented in Section IV-B1. The PPGA's implementation of the initial solutions generated by the trained ANNs to produce a new schedule is described in Section IV-B2.

1) INITIAL SOLUTIONS RESULTED BY THE ANNs

In *Case 2*, the ANNs are used to generate initial solutions for PPGA when machine failure situations during production differ from the inputs in the training process. To obtain initial solutions for a given machine failure situation, the processing times of jobs in each machine are passed through the corresponding ANNs (as shown in Fig. 13).

Each ANN generates confidences, $\hat{s}_{jk}$, for a specific job (j) in each position (k) in the sequence, which represent the level of suitability for assigning the job to that position. However, this approach may result in the occurrence of repetitive positions in the sequence as multiple jobs can have the highest confidences in the same position. This repetition of positions violates the condition of flow shop production, where each job should be assigned to a unique position.

In the presented example, feeding a new machine failure situation that deviates from the machine failure scenario in the training process through the trained ANNs results in multiple jobs (i.e., job 1, job 2, job 3, and job 4) being assigned to position 1, leading to repetitive positions for multiple jobs.

To address the issue, a process for managing repetitive positions is proposed as depicted in Fig. 14. In Step 1, the confidences of jobs for each position are sorted in descending order. In Step 2, for each position, starting from the first to the last position, the job having the highest confidence is assigned to that position while adhering to the following conditions: 1) Jobs that have already been assigned to other positions cannot be used, and 2) If the difference in confidences between selected jobs is less than δ (which is set as 0.03 in this research), these jobs will be considered as having the same priority for position assignment, potentially leading to multiple solutions.

The solutions provided from the ANNs are included in the initial population of the PPGA. To preserve diversity in the search space on PPGA, the maximum number of initial solutions from the ANNs is restricted to half of the total population size ($P/2$). However, if the total number of the solutions from the ANNs exceeds this limit due to repetitive positions, the solutions are selected by their makespan (C).

2) IMPLEMENTING RESCHEDULING KNOWLEDGE FROM THE ANNs INTO THE PPGA

The utilization of PPGA in the knowledge implementation phase allows for addressing new instances that differ from the inputs used to train the ANNs. The process of PPGA in this phase is similar to the knowledge creation phase, with the difference being that the initial population incorporates the initial solutions obtained from the trained ANNs, which store the rescheduling knowledge from various machine failure scenarios, rather than relying purely on a randomized initial population. By incorporating the knowledge from the trained ANNs, the proposed PPGA can generate new schedules for

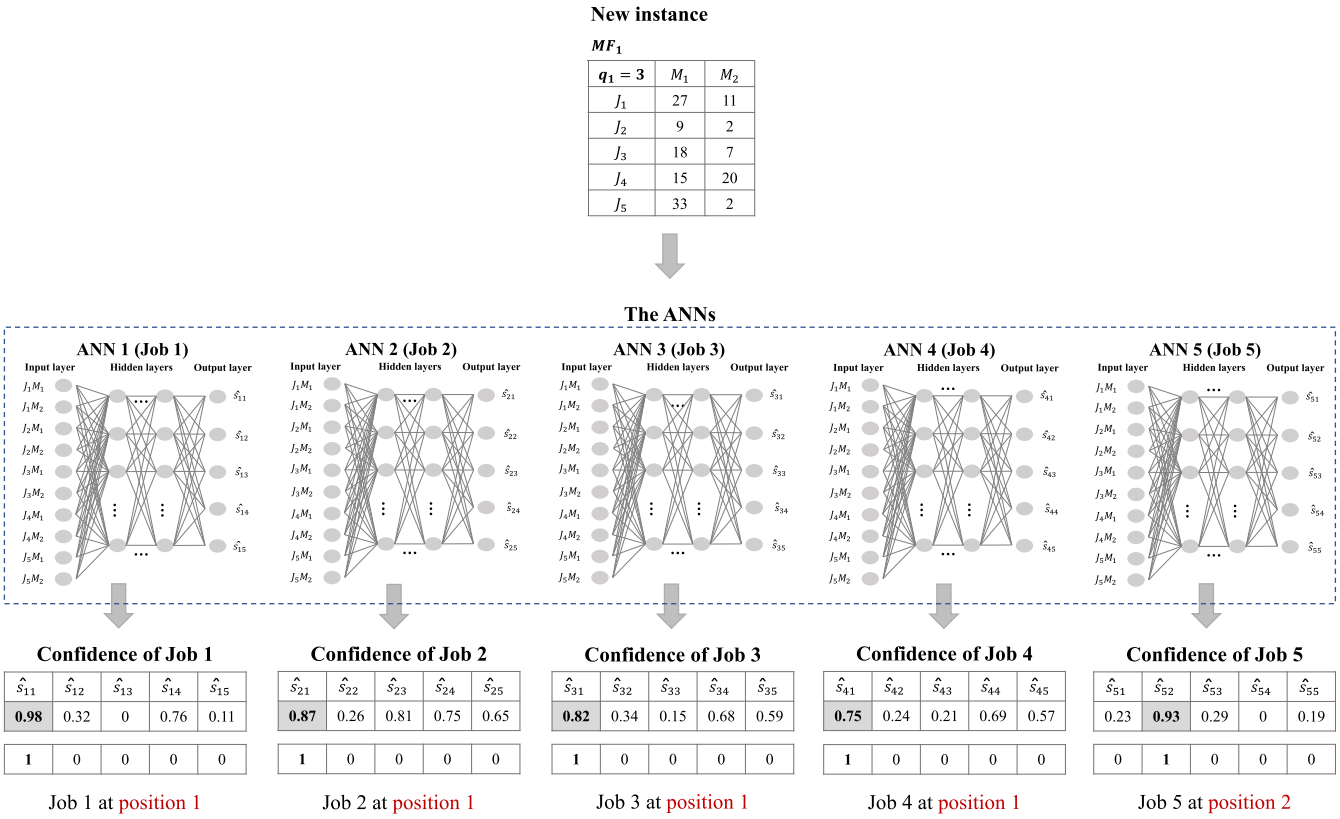


FIGURE 13. Example of repetitive positions in the sequence.

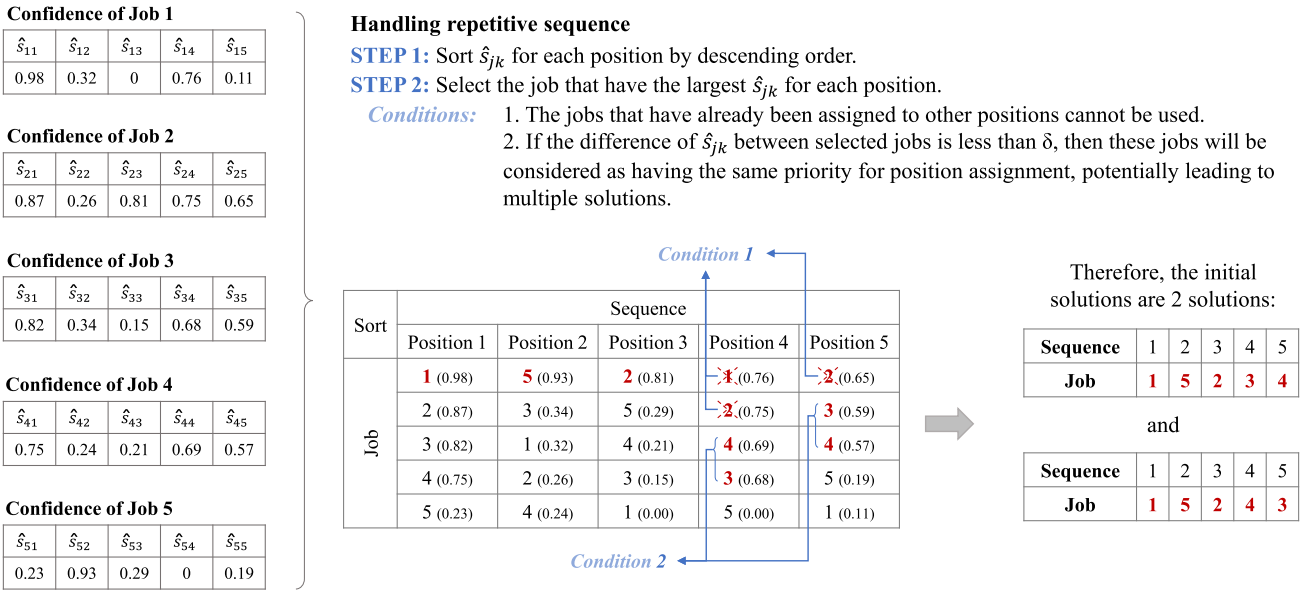


FIGURE 14. Handling repetitive positions in the sequence.

machine failure occurrences, potentially improving computational time.

V. EXPERIMENTAL ANALYSIS

In this research, the proposed PPGA-ANNs framework (Fig. 1) is developed for a rapid rescheduling of large-scale flow shop production with machine failure occurring. All experiments for the proposed framework are implemented using the Python programming language and facilitated by the Anaconda platform. The experiments are conducted on a laptop computer with a 16 GB RAM and 11th Gen Intel(R) Core(TM) i5 CPU clocked at 2.4 GHz.

The following sections are conducted to demonstrate the experiments of the proposed PPGA-ANNs framework. The generation of machine failure scenarios in flow shop production is introduced in Section V-A. In Section V-B, we present the details of the adjustment of hyperparameters used in the standard GA, the PPGA, and the ANNs. Finally, in Section V-C, the proposed PPGA-ANNs framework is evaluated by comparing the performance with other approaches and presenting the result analysis.

A. MACHINE FAILURE SCENARIOS GENERATION

The machine failure scenarios are tested with the benchmark data set designed by Taillard [7], widely applied in many studies, especially in scheduling problems of different scales. This research adopts the processing time of large-scale flow shop production with 20 jobs ($n = 20$) and 10 machines ($m = 10$) in the benchmark's first data set. The processing time (p_{ij}) takes a uniform distribution value on $U[1,99]$.

To simulate the situation when machines fail in production, the scenarios are composed by (1). Consequently, we begin with a base instance from the benchmark, where the processing time of all jobs is uniformly distributed from 1 to 99 minutes. We set q_i to four values for this experiment: 1.5, 2, 2.5, and 3, by assuming that the limit of changing processing time (Q) acceptable by the individual manufacturer is less than or equal to 3.

Based on the above parameter setting, the number of scenarios is $4 \times (2^{10} - 1)$ plus one case of the ordinary situation without machine failure, which results in 4,093 scenarios in total. Each scenario is solved by the proposed PPGA. The ANNs are then trained using the scenarios of the disturbed processing time as the inputs and the results from PPGA as the target outputs. The ANNs will provide an excellent solution if the training instances can accurately simulate the disturbed production.

B. HYPERPARAMETERS ADJUSTMENT

The present research defines the hyperparameters for addressing rescheduling of the flow shop production problem with 20 jobs and 10 machines. Experimental investigations are conducted to determine the optimal values for these hyperparameters. The hyperparameters used in the standard GA, the proposed PPGA, and the ANNs are detailed in Table 2.

Results from the experimentation reveal that for the proposed PPGA, a population size (P) of 300 chromosomes is suitable for the considered problem. Fig. 15 shows the experiment with base case of the scheduling problem, where the PPGA obtains a relatively short runtime for a population size of less than 300 chromosomes, but it has a longer makespan (C). On the other hand, for population sizes exceeding 300 chromosomes, the algorithms require a longer runtime and a longer makespan (C).

Moreover, the literature review indicates that a higher crossover probability (Cr) accelerates the convergence to a solution at the expense of reduced population diversity [50]. A higher mutation probability (Mu) enhances search space exploration but increases the likelihood of encountering in a suboptimal solution [50]. Hence, the proposed PPGA uses a crossover probability of 0.8 and a mutation probability of 0.01.

In addition, the maximum number of iterations (Itr) and the maximum runtime ($Mrun$) are the termination criteria for PPGA. The experiment reveals that 500 iterations are suitable as the PPGA converges prematurely before reaching the optimal solution with the number of iterations less than 500. In contrast, for a number of iterations exceeding 500, the algorithms exhibit prolonged computational time even after already discovering the optimal solution. As for the maximum runtime ($Mrun$), it is assumed that a duration exceeding 10 minutes does not meet the rapid rescheduling requirement.

TABLE 2. The hyperparameters used in the standard GA, the PPGA, and the ANNs.

Model	Hyperparameters	Values
Standard GA	Population size (P)	300 chromosomes
	Crossover probability (Cr)	0.8
	Mutation probability (Mu)	0.01
	Number of iterations (Itr)	500 iterations
	Maximum runtime ($Mrun$)	10 minutes
PPGA	Population size (P)	300 chromosomes
	Crossover probability (Cr)	0.8
	Mutation probability (Mu)	0.01
	Number of iterations (Itr)	500 iterations
	Maximum runtime ($Mrun$)	10 minutes
	Percentage of improvement (γ)	0.1
	Number of compared iterations (β)	50 iterations
	Percentage of selection the best chromosomes (α) and randomly creating new chromosomes ($100 - \alpha$)	25 and 75
	Number of perturbations (Ptb)	5 perturbations
ANNs	Hidden layers	5 layers
	Hidden nodes	1st layer: 70 nodes 2nd layer: 60 nodes 3rd layer: 50 nodes 4th layer: 40 nodes 5th layer: 30 nodes
	Activation function of hidden layers	ReLU activation
	Activation function of output layer	Sigmoid activation
	Learning rate	0.001
	Epochs	2000, 3000, 4000, 5000
	Batch size	32
	Optimizer	Adam

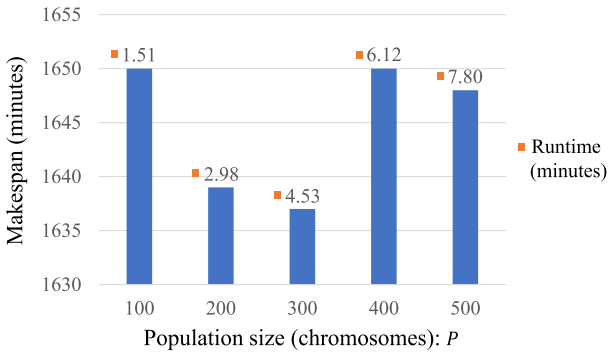


FIGURE 15. Experimentation on population size (P).

The perturbation operation stage of the PPGA is triggered when the best chromosome's fitness value in the current iteration has not improved by more than 0.1% (γ) compared to the best fitness value in the previous 50 iterations (β). The percentage improvement threshold (γ) and the number of considered previous iterations (β) should be defined based on the user's or individual manufacturer's preferences. Moreover, the improvement threshold (γ) does not have to be in percentage form and may be in minutes of the makespan (C) or any other appropriate unit, depending on the decision of individual manufacturers.

The top 25% of the chromosomes in the population (α), based on their fitness values, are then selected to preserve the best chromosomes from the standard genetic operation stage. The population's remaining 75% ($100-\alpha$) is produced randomly to increase population diversity and investigate uncharted search spaces.

According to the maximum number of perturbations (Ptb), the PPGA continues to perform the perturbation operation until it reaches five times. However, in some cases, the maximum number of iterations (Itr) can be reached before the maximum number of perturbations (Ptb).

The hyperparameters used in the ANNs are selected from a range of values to determine the optimal combination of parameters that can achieve high performance and improve the accuracy and robustness of the ANNs. Table 3 shows the range of values for each hyperparameter, which is consistently applied to all ANNs. The number of hidden layers is tested between 1 and 7 layers, and the number of nodes in each hidden layer is set within the range of n and $n \times m$, where n represents the number of output nodes (number of jobs), m represents the number of machine, and $n \times m$ represents the number of input nodes. The learning rate is tested at 0.001, 0.01, and 0.1, while the number of epochs is varied from 1000 to 5000. The batch size is selected from either 32 or 64. Moreover, two optimization algorithms, Adam and stochastic gradient descent (SGD), are also tested in the ANNs.

This research conducts a comprehensive exploration of the components of ANNs, involving trial and error experimentation with various hyperparameters, including the optimizer, learning rate, and batch size. Subsequently, a detailed ablation study is conducted to investigate the impact of different configurations of the hidden layers. The initial experimentation focuses on determining the optimal number of hidden layers, starting with the ANN for job 1 using a baseline of two layers and training it for 1000 epochs. Further analysis explores the influence of the number of nodes in each hidden layer. Remarkably, once the ANN for job 1 achieves an accuracy exceeding 99%, the configuration of the number of hidden layers and the number of nodes in each hidden layer is finalized. For the remaining jobs, optimization is achieved by varying the number of epochs. This ablation study provides valuable insights into the optimal hyperparameter settings and configuration of the ANNs, enabling the generation of accurate and efficient rescheduling solutions in the production environment. However, it is important to acknowledge that this specific ablation study may not be directly transferable to other problems or data sets. The efficacy of hyperparameter configurations is highly dependent on the unique characteristics and complexities of individual problem domains and data sets.

TABLE 3. The range of hyperparameters in the ANNs.

Hyperparameters	Min	Max	Step
Hidden layers	1	7	6
Hidden nodes	20	200	18
Learning rate	0.001	0.1	2
Epochs	1000	5000	4
Batch size	32	64	1
Optimizer	Adam, SGD		

The ANNs are constructed based on the processing time data for 20 jobs ($n = 20$) and 10 machines ($m = 10$), resulting in 20 separate networks comprising 200 input and 20 output nodes. Based on the experiment, the suitable number of hidden layers is 5 layers with 70, 60, 50, 40, and 30 hidden nodes, respectively. The hidden layers are equipped with the Rectified Linear Unit (ReLU) activation function. The output layer uses a sigmoid function, which maps outputs to a range between 0 and 1, making it ideal for binary values that need to be interpreted as probabilities. The learning rate is set to 0.001. The number of epochs ranges from 2000 to 5000 to achieve high accuracy. Based on the experiment results, the optimal number of epochs for each job's the ANNs are determined. The ANNs for job 1, 2, 4, 5, and 16 achieve high accuracy within 2000 epochs, while those for jobs 3, 6, 7, 9, 10, 12, 13, 17, and 18 require 3000 epochs. The ANNs for job 8, 11, 14, and 19 use 4000 epochs, and those for jobs 15 and 20 use 5000 epochs. The batch size used in the ANNs is 32. The optimizer in the ANNs is Adam, a popular algorithm variant of stochastic gradient descent optimization. The Adam optimizer is applied as one of the arguments when compiling the ANNs. It uses moving averages of the parameters to provide a running estimate of the gradients and helps to maintain a stable learning rate and avoid oscillations. It is computationally efficient and well-suited for large-scale problems [54].

The values reported in Table 2 represent hyperparameters critical in controlling the learning process. Typically, these values are determined through an iterative trial and error search process. Although the same hyperparameter values may be suitable for other problems, it is necessary to periodically explore alternative values to optimize the algorithm's performance.

C. EVALUATION EXPERIMENTS

This section presents an evaluation of the proposed PPGA-ANNs framework by demonstrating the capabilities of its two main components, the PPGA and the ANNs. The PPGA's performance is assessed by comparing its results with those of existing algorithms. The benefit of the perturbation operation stage in avoiding local optima can be revealed

TABLE 4. Comparison of the makespan between PPGA, standard GA, HGA, IIGA, DSOMA, and DWWO.

Problem	$n \times m$	Upper bound	PPGA	Standard GA	HGA	IIGA	DSOMA	DWWO
Ta011	20x10	1582	**1625**	1642	1955	2011	1698	2044
Ta021	20x20	2297	**2399**	2405	2912	2973	2436	2973
Ta031	50x5	2724	**2740**	2774	3127	3161	3033	3170
Ta041	50x10	3037	**3251**	3256	4251	4274	3638	4274
Ta051	50x20	3886	**4253**	**4253**	6138	6129	4511	6129
Ta061	100x5	5493	**5685**	**5685**	6492	6397	6151	6433
Ta071	100x10	5776	**6113**	6156	8115	8077	7042	8093
Ta081	100x20	6330	**6971**	7046	10745	10700	7854	10727
Ta091	200x10	10872	**11328**	11423	15739	15319	13507	15418
Ta101	200x20	11393	**12489**	12684	20148	19681	15027	19724

through a comparison of the scheduling results obtained from the PPGA and the standard GA. Moreover, we evaluate the effectiveness of applying the knowledge stored in the ANNs for rapid rescheduling by comparing the convergence of our proposed PPGA-based initial solutions from the trained ANNs with the PPGA and the standard GA.

1) COMPARISON OF THE PPGA IN THE KNOWLEDGE CREATION PHASE WITH EXISTING ALGORITHMS

In the knowledge creation phase of the proposed PPGA-ANNs framework, the PPGA, a novel approach to improve the standard GA for production scheduling by incorporating a perturbation operation stage, is developed. The PPGA aims to optimize the production scheduling performance by minimizing the makespan criterion.

To evaluate the effectiveness of the PPGA in terms of makespan, this research utilizes the comprehensive benchmark developed by Taillard [7] that includes instances of different scales. The PPGA is run five times, and the best makespan values are recorded to compare the performance with the standard GA, hybrid GA (HGA) [55], improved iterated greedy algorithm (IIGA) [56], discrete water wave optimization algorithm (DWWO) [57], and upper bounds (the minimum possible makespan) [7].

Table 4 shows the makespan results obtained from different approaches using the same data set from the benchmark. It is noteworthy that the PPGA utilized for solving problems of various scales does not incorporate the criterion of maximum runtime (*Mrun*), as the objective is to obtain the best solution for comparison with existing approaches. Therefore, the PPGA was executed until it reached the specified number of iterations (*Itr*). The results indicate that the proposed PPGA outperforms other algorithms, yielding better makespan values. In some cases, the standard GA can achieve a near-optimal solution similar to the PPGA, indicating that the standard GA does not fall into local optima.

Moreover, the perturbation operation stage in the PPGA is evaluated to determine its efficiency in enhancing population diversity and increasing the chance of finding optimal solutions. This stage preserves the best chromosomes from the previous iteration while randomly generating new chromosomes. The evaluation involves comparing the convergence performance of the PPGA with that of the standard GA using the same random seed to solve flow shop scheduling with machine failure scenarios. The results are shown in Fig. 16.

The experiment illustrated examples of four extreme machine failure scenarios: (a) Machine 1, 5, and 9 ($i = \{1, 5, 9\}$) fail with $q_i = 3$, (b) Machine 2, 4, 6, 8, and 10 ($i = \{2, 4, 6, 8, 10\}$) fail with $q_i = 3$, (c) Machine 3, 4, 5, 6, 7, 8, and 9 ($i = \{3, 4, \ldots, 9\}$) fail with $q_i = 3$, and (d) Machine 1, 2, 3, 4, 5, 6, 7, 8, and 9 ($i = \{1, 2, \ldots, 9\}$) fail with $q_i = 3$. In the graph, the red points represent the number of perturbations (*Ptb*) performed during the perturbation operation stage. The results indicate that the PPGA which incorporates the perturbation operation stage with the standard genetic operation stage, provides lower makespan (C) of better near-optimal solutions compared to the standard GA. Specifically, the convergence curves of the PPGA exhibited improvements even in extreme cases, demonstrating the efficiency of the perturbation operation stage in exploring uncharted search spaces.

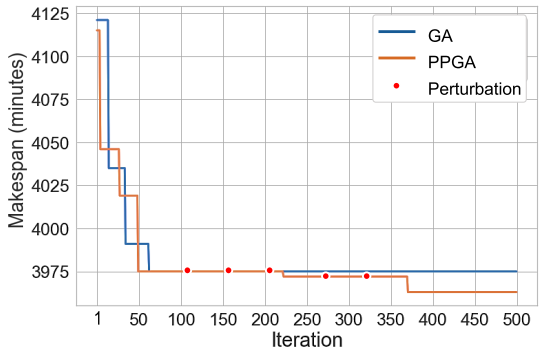
Additionally, the box plot (Fig. 17) provides a visual representation of the comparison between the standard GA and PPGA in terms of the makespan, utilizing the data set from Ta011 of Tailard's benchmark. The box plot summarizes the distribution of the makespan values obtained from ten experimental runs.

The box in the plot represents the interquartile range (IQR), encompassing the lower quartile (Q1) to the upper quartile (Q3) values. The median value is indicated by the horizontal line inside the box. The whiskers extend from the box to denote the minimum and maximum values within the range.

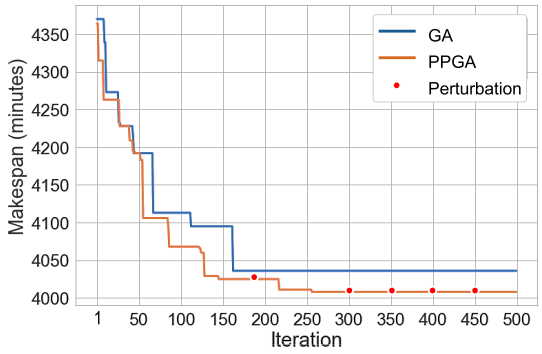
Upon analyzing the box plot, the standard GA exhibits a median makespan of 1649.5 minutes, whereas the PPGA demonstrates a median makespan of 1634.5 minutes. These median values provide insights into the central tendency of each algorithm's performance. The median makespan value for the PPGA is lower than that of the standard GA, offering compelling evidence of the PPGA's superior makespan results. This finding underscores the effectiveness of the proposed PPGA in the context of production scheduling and highlights its potential for improving production efficiency and resource utilization.

2) EFFECTIVENESS OF THE PROPOSED FRAMEWORK IN TERMS OF COMPUTATIONAL TIME

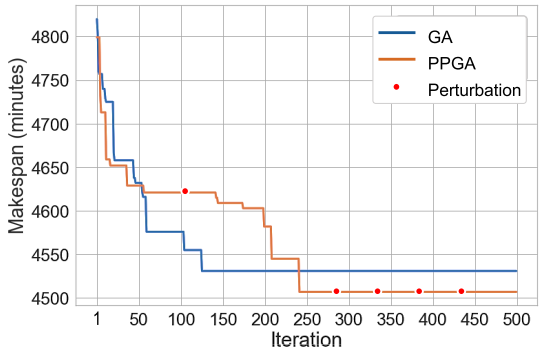
The PPGA-ANNs framework proposes the use of the ANNs to store the knowledge from PPGA in the knowledge creation



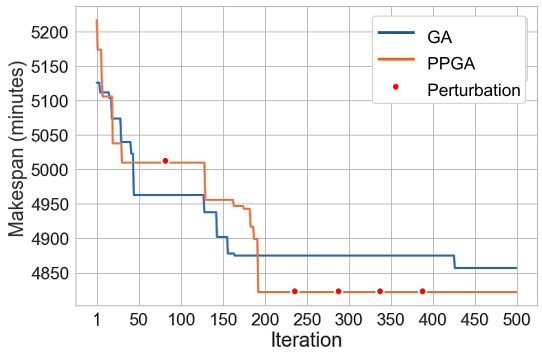
(a) Machine 1, 5, and 9 fail with $q_i = 3$



(b) Machine 2, 4, 6, 8, and 10 fail with $q_i = 3$



(c) Machine 3, 4, 5, 6, 7, 8, and 9 fail with $q_i = 3$



(d) Machine 1, 2, 3, 4, 5, 6, 7, 8, and 9 fail with $q_i = 3$

FIGURE 16. Comparison of the result between the standard GA and the PPGA.

phase. In the knowledge implementation phase, the ANNs are then used to provide direct solutions or initial solutions for the PPGA to address the issue of machine failures that could result in unfavorable solutions for primary sequence execution.

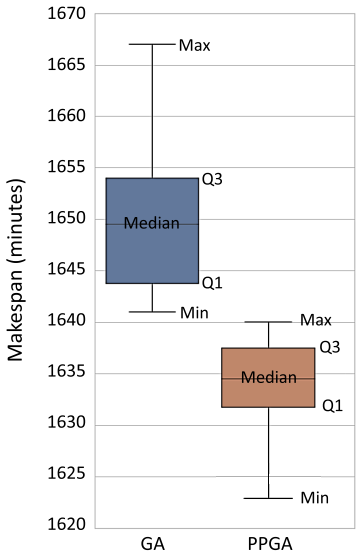


FIGURE 17. The box plot of the makespan (minutes) between GA and PPGA.

The performance of the ANNs is evaluated based on the loss and accuracy metrics. The numerical results demonstrate the ANNs strong memory capabilities and excellent understanding of relationships within the network, as shown by the high accuracy scores and low loss values. The ANNs exhibit an average accuracy score of 99.85% and an average loss of 0.37%, suggesting the high potential in mitigating machine failures and improving the performance of the PPGA.

The ANNs aim to generate initial solutions that can expedite the PPGA in searching for optimal solutions and allow for rapid production rescheduling, even when the actual machine failure situation differs from the training inputs.

To evaluate the effectiveness of the initial solutions generated by the ANNs in the PPGA-ANNs, new instances are passed through the ANNs. These new instances differ from the machine failure scenarios used during training, as we introduce a percentage change in the processing time of the inputs. The percentage change formulation is expressed by the following equation:

$$Percentage\ change = \frac{|q_{i_{new}} - q_i|}{q_i} \times 100 \qquad (7)$$

where q_i represents the impact of machine failure on processing time in the training scenarios, $q_{i_{new}}$ represents the new impact of machine failure on processing time that has not been used in the training process, and the absolute value ensures that the resulting percentage change value is positive regardless of the direction of change.

The resulting percentage change value indicates the magnitude of the increase or decrease in processing time in the new instance compared to the inputs used during the training process. This allows us to evaluate the effectiveness of the initial solutions and the potential to accelerate the rescheduling

process in a broader range of situations beyond the machine failure scenarios used in the training process.

The PPGA-ANNs framework is tested using fifteen instances, as depicted in Fig. 18. These instances include: (a) three instances that are identical to the inputs used during training process (with a percentage change of 0%). Furthermore, (b) three instances are generated with a 10% increase in the percentage change of the new instances ($q_{i_{new}} = 3.3$) from the inputs ($q_i = 3$). Additionally, (c) three instances are generated with a 30% increase in the percentage change of the new instances ($q_{i_{new}} = 3.9$) from the inputs ($q_i = 3$). Another set of (d) three instances are generated with a 50% increase in the percentage change of the new instances ($q_{i_{new}} = 4.5$) from the inputs ($q_i = 3$). Finally, (e) three instances are generated with a 100% increase in the percentage change of the new instances ($q_{i_{new}} = 6$) from the inputs ($q_i = 3$). The processing times of the testing instances are normalized using the mean ($\overline{p_{ij}}$) and standard deviation (σ_{ij}) from the training set to ensure consistency in the evaluation process.

Fig. 18 shows the convergence behavior of the standard GA, the PPGA, and the PPGA-ANNs with respect to the percentage change of the new instances. The comparisons among instances can provide insights into the potential benefits of incorporating initial solutions generated by the ANNs in enhancing overall performance.

The results reveal that the GA fails to achieve optimal solutions for all new instances. For instances with 0% deviation from the inputs, i.e., (a1), (a2), and (a3), the initial solutions provided by the trained ANNs significantly assist the PPGA, which leads to PPGA-ANNs achieving the optimal solutions at the first iteration.

The PPGA-ANNs framework also shows improved convergence speed compared to PPGA for new instances with 10% and 30% deviation from the inputs. For instances (b1), (b2), and (c1), finding the optimal solutions is not too complex; therefore, PPGA-ANNs can easily achieve the optimal solutions with a few iterations. Moreover, for instances (b3), (c2), and (c3) where finding the optimal solutions is more complex, the PPGA-ANNs can still outperform PPGA in terms of providing the optimal solutions faster.

For instances with a deviation of more than 50% from the inputs, the initial solutions provided by the trained ANNs are found to be ineffective. In the case of instances (d1) and (e1), the solutions obtained from the standard GA, PPGA, and PPGA-ANNs are all the same because permuting the sequence of jobs is not effective when the new impact of machine failure on processing time ($q_{i_{new}}$) is too significant and only one machine has failed. For instances (d2), (d3), (e2), and (e3), the runtime of PPGA-ANNs cannot be guaranteed, even when the initial solutions from the trained ANNs are used. However, PPGA-ANNs can still find optimal solutions that are better than those obtained by the standard GA.

The results imply that the trained ANNs can effectively provide initial solutions for the new instances with a deviation of 30% or less. As explained in Sections IV-A1, the acceptable impact on processing time (q_i) in this research is confined to a range that does not require production to be stopped. Therefore, the initial solutions from the ANNs can be employed to the PPGA for rescheduling the previous schedules when the deviation of processing time falls within this acceptable range, and can achieve faster results than the PPGA.

However, it is important to acknowledge the limitations of the generalizability of the trained ANNs to completely new and unknown instances. The performance of the ANNs can be negatively impacted when confronted with instances that significantly deviate from the training data set. In this research, the initial solutions generated by the ANNs are utilized as the population for the first iteration of PPGA. These initial solutions, illustrated by the first iteration of the green line (PPGA-ANNs), represent the solution with respect to the makespan aspect of the problem.

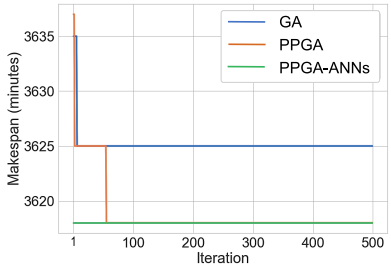
In the first iteration of each green line, it is observed that when a new instance precisely aligns with the given inputs ((a)), the ANNs are capable of accurately predicting the outcome and yielding the optimal solution. However, when the new instance deviates from the given inputs ((b), (c), (d), and (e)), the initial solution generated by the ANNs may occasionally be suboptimal. In such cases, the complementary nature of the PPGA comes into effect. By employing the PPGA-ANNs framework, we can effectively harness the strengths of both algorithms to obtain solutions within a faster computational time.

VI. DISCUSSION

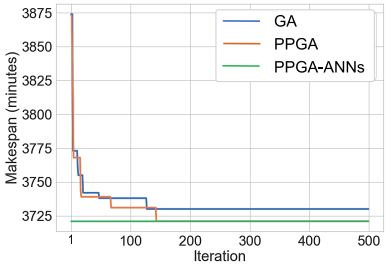
In practical applications, it is suggested to process the PPGA and the PPGA-ANNs framework in parallel. This parallel processing approach is advocated due to the inherent uncertainty surrounding the deviation or percentage change in the new impact of machine failure on processing time ($q_{i_{new}}$) with respect to the provided inputs. By simultaneously running both algorithms, it guarantees the generation of optimal solutions that surpass the performance of the standard GA.

Either PPGA or PPGA-ANNs for achieving optimal solutions within a shorter computational time depends on the particular instance under consideration such as the problem's complexity, the unique characteristics of the production environment, and the attributes of the disturbance encountered. By executing both algorithms in parallel, manufacturers can capitalize on the respective strengths of each approach, enabling them to obtain the most advantageous rescheduling solutions.

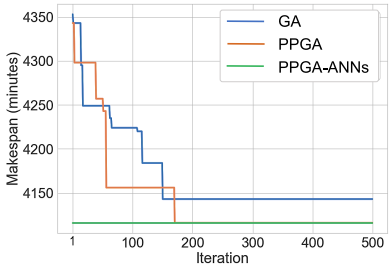
Moreover, to assess the importance of rescheduling, we conducted a research comparing makespan (C) for a production process with and without rescheduling by PPGA-ANNs under various machine failure scenarios, ranging from the failure of one machine to the failure of all ten machines. In each scenario, we randomly selected three sets of failed machines from all possible sets of failed machines under a specific number of failed machines. The makespans from the three sets of failed machines are then averaged.



(a1) Machine 1 fails with $q_{i_{new}} = 3$

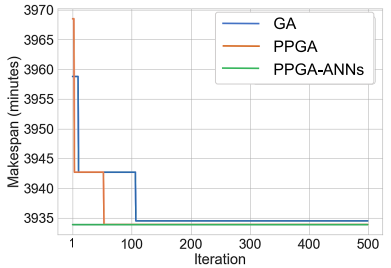


(a2) Machine 2, 6, and 8 fail with $q_{i_{new}} = 3$

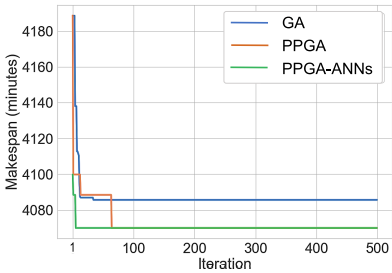


(a3) Machine 1, 3, 5, 7, and 9 fail with $q_{i_{new}} = 3$

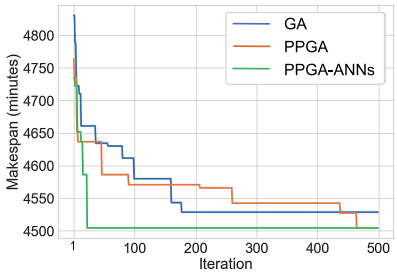
(a) The percentage change between the new instances and the inputs is 0%



(b1) Machine 1 fails with $q_{i_{new}} = 3.3$

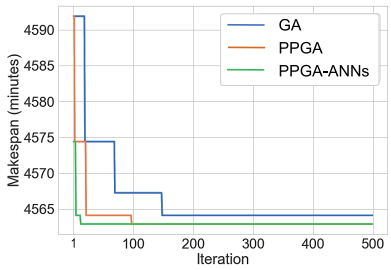


(b2) Machine 2, 6, and 8 fail with $q_{i_{new}} = 3.3$

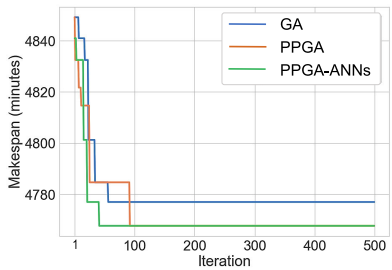


(b3) Machine 1, 3, 5, 7, and 9 fail with $q_{i_{new}} = 3.3$

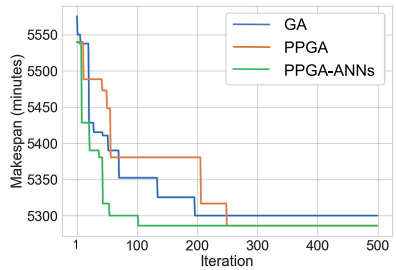
(b) The percentage change between the new instances and the inputs is 10%



(c1) Machine 1 fails with $q_{i_{new}} = 3.9$

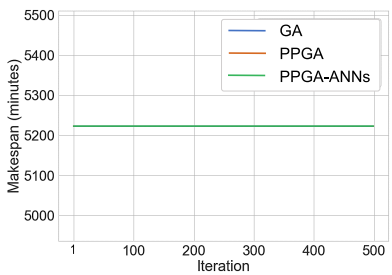


(c2) Machine 2, 6, and 8 fail with $q_{i_{new}} = 3.9$

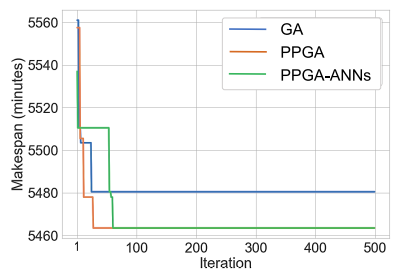


(c3) Machine 1, 3, 5, 7, and 9 fail with $q_{i_{new}} = 3.9$

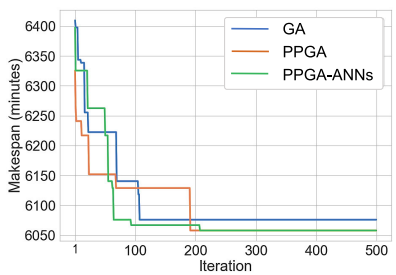
(c) The percentage change between the new instances and the inputs is 30%



(d1) Machine 1 fails with $q_{i_{new}} = 4.5$

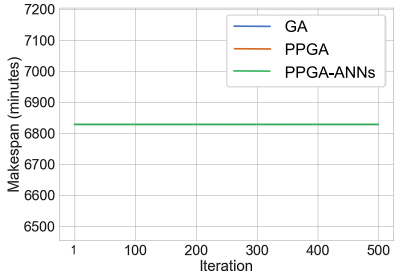


(d2) Machine 2, 6, and 8 fail with $q_{i_{new}} = 4.5$

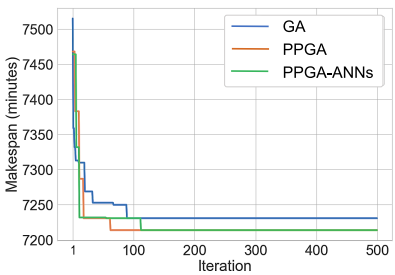


(d3) Machine 1, 3, 5, 7, and 9 fail with $q_{i_{new}} = 4.5$

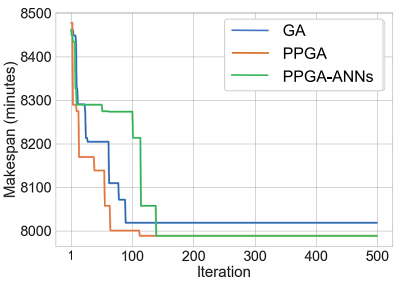
(d) The percentage change between the new instances and the inputs is 50%



(e1) Machine 1 fails with $q_{i_{new}} = 6$



(e2) Machine 2, 6, and 8 fail with $q_{i_{new}} = 6$



(e3) Machine 1, 3, 5, 7, and 9 fail with $q_{i_{new}} = 6$

(e) The percentage change between the new instances and the inputs is 100%

FIGURE 18. Comparison of the convergence between the standard GA, PPGA, and PPGA with initial solutions from the trained ANNs (PPGA-ANNs).

TABLE 5. Comparison of the average makespan (minutes) between un-rescheduled (*Un*) and rescheduled (*Re*) production when machine failure occurs.

No. failed machines	$q_i = 1.5$			$q_i = 2$			$q_i = 2.5$			$q_i = 3$		
	Un	*Re*	Minutes improve	*Un*	*Re*	Minutes improve	*Un*	*Re*	Minutes improve	*Un*	*Re*	Minutes improve
1	1991.67	1910.67	81.00	2483.00	2418.33	64.67	3000.00	2936.00	64.00	3530.00	3462.67	67.33
2	2040.33	1934.33	106.00	2559.33	2457.33	102.00	3086.33	2988.00	98.33	3627.67	3526.00	101.67
3	2172.67	2023.33	149.33	2751.67	2578.33	173.33	3338.00	3128.67	209.33	3934.67	3706.67	228.00
4	2241.67	2133.67	108.00	2877.67	2741.67	136.00	3510.33	3303.00	207.33	4153.33	3910.33	243.00
5	2297.33	2210.33	87.00	2978.67	2867.67	111.00	3660.33	3537.33	123.00	4353.00	4154.33	198.67
6	2329.00	2193.67	135.33	3074.67	2819.00	255.67	3811.33	3473.67	337.67	4557.00	4125.67	431.33
7	2401.33	2298.67	102.67	3173.67	2998.00	175.67	3936.67	3750.67	186.00	4709.00	4452.00	257.00
8	2431.33	2319.67	111.67	3228.00	3066.00	162.00	4014.67	3781.33	233.33	4811.33	4543.00	268.33
9	2453.33	2389.67	63.67	3272.00	3183.67	88.33	4079.67	3962.33	117.33	4898.33	4715.67	182.67
10	2463.00	2463.00	0.00	3282.00	3282.00	0.00	4099.00	4099.00	0.00	4928.00	4928.00	0.00

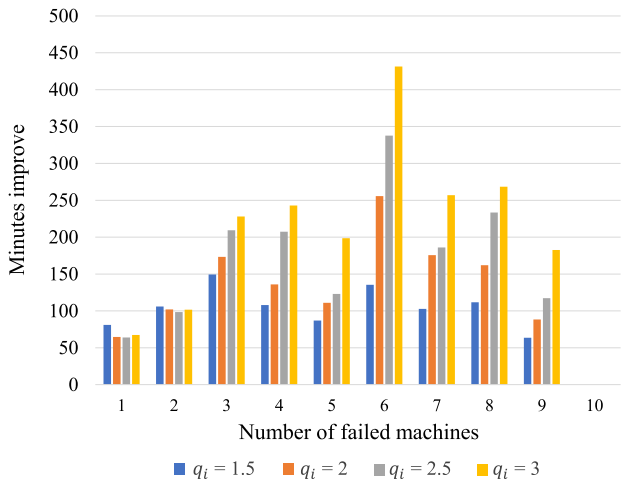


FIGURE 19. The improvement in minutes of the average makespan between un-rescheduled (*Un*) and rescheduled (*Re*) production when machine failure occurs.

As shown in Table 5, the results show that the implementation of rescheduling (represented by *Re*) resulted in the shorter average makespans compared to the un-rescheduled production (represented by *Un*) in all scenarios. Moreover, the runtime for the rescheduling process was between 214-377 seconds (less than 6 minutes). These findings demonstrate the significance of rescheduling in improving production efficiency by reducing makespans within a short computational time.

The results in Fig. 19 plotted based on the data in Table 5, indicate that rescheduling effectively improves the makespans in almost scenarios. For scenarios where only one or two machines fail, the improvement in the makespans is consistent regardless of the impact of processing time (q_i) when a machine fails. However, when three to nine machines fail, the makespans show significant improvement with higher values of q_i, particularly in the case of six failed machines.

In contrast, in a scenario where all ten machines fail, the improvement in makespan is found to be insignificant. This is because the impact of machine failure (q_i) is uniformly distributed among all machines, resulting in no significant improvement in makespan through rescheduling. Although the experiment indicates that rescheduling is most urgent when six machines fail, it may not necessarily hold true for other data sets.

VII. CONCLUSION AND FUTURE WORK

Effective production scheduling and rescheduling strategies are crucial in today's highly competitive marketplace, where unexpected events can occur at any time, making production rescheduling a critical aspect of decision-making in dynamic manufacturing environments. To address this challenge, we propose a novel framework that combines the Perturbation Population Genetic Algorithm (PPGA) with Artificial Neural Networks (ANNs). This integration leverages the strengths of both meta-heuristic and supervised learning approaches, enabling the framework to effectively generate new schedules in a timely manner while minimizing makespan.

The proposed PPGA-ANNs framework and its methodology offer a dependable solution for production rescheduling in flow shop environments with machine failure occurrence. Its superiority in terms of makespan and computational time has been demonstrated through comprehensive comparative experiments. The proposed framework's capability comes from two key components: the PPGA in the knowledge creation phase and the ANNs in the knowledge implementation phases. The PPGA contributes to better performance in reducing makespan, while applying the initial solutions generated by the trained ANNs to the PPGA provides shorter runtime for obtaining a new schedule.

In conclusion, this research contributes to the body of knowledge in flow shop rapid production rescheduling with machine failure problems. The proposed framework and its methodology minimize the makespan with short computational time, offering a reliable and flexible solution for real-world production. Consequently, the proposed PPGA-ANNs framework can be applied as an alternative for production rescheduling to address machine failures, particularly when multiple machines are involved, for improving the efficiency and performance of manufacturing processes.

To further enhance the realism and relevance of the problem and provide possible extensions for future work, the

scope of the problem can be expanded to include additional complexities. This can involve increasing the number of jobs and machines, or extending the analysis to include different production environments, such as job shop production, for a more comprehensive representation of real-world manufacturing situations. Additionally, a termination criterion that stops the PPGA if it fails to find a better solution within a certain number of iterations, such as 100 iterations, can be proposed to reduce the computational time required for rescheduling.

Additionally, the performance of the proposed PPGA-ANNs framework is subject to the scenarios used to represent disturbed situations. To overcome this limitation and improve the applicability of the proposed framework, it is recommended to integrate reinforcement learning techniques [58] such as Q-learning. By doing so, the algorithm can store and manage novel knowledge or unseen instances that differ significantly from the initial scenarios, thereby enhancing the algorithm's adaptability.

REFERENCES

[1] M. Ghaleb, H. Zolfagharinia, and S. Taghipour, "Real-time production scheduling in the Industry-4.0 context: Addressing uncertainties in job arrivals and machine breakdowns," *Comput. Oper. Res.*, vol. 123, Nov. 2020, Art. no. 105031.

[2] I. R. Uhlmann and E. M. Frazzon, "Production rescheduling review: Opportunities for industrial integration and practical applications," *J. Manuf. Syst.*, vol. 49, pp. 186–193, Oct. 2018.

[3] G. E. Vieira, J. W. Herrmann, and E. Lin, "Rescheduling manufacturing systems: A framework of strategies, policies, and methods," *J. Scheduling*, vol. 6, no. 1, pp. 39–62, Jan. 2003.

[4] A. Dutta, "Reacting to scheduling exceptions in FMS environments," *IIE Trans.*, vol. 22, no. 4, pp. 300–314, Dec. 1990.

[5] S. Lu, J. Pei, X. Liu, and P. M. Pardalos, "A hybrid DBH-VNS for high-end equipment production scheduling with machine failures and preventive maintenance activities," *J. Comput. Appl. Math.*, vol. 384, Mar. 2021, Art. no. 113195.

[6] Z. Jiang, S. Yuan, J. Ma, and Q. Wang, "The evolution of production scheduling from Industry 3.0 through Industry 4.0," *Int. J. Prod. Res.*, vol. 60, no. 11, pp. 3534–3554, Jun. 2022.

[7] E. Taillard, "Benchmarks for basic scheduling problems," *Eur. J. Oper. Res.*, vol. 64, no. 2, pp. 278–285, Jan. 1993.

[8] S. C. Graves, "A review of production scheduling," *Oper. Res.*, vol. 29, no. 4, pp. 646–675, 1981.

[9] J. Berlińska and B. Przybylski, "Scheduling for gathering multitype data with local computations," *Eur. J. Oper. Res.*, vol. 294, no. 2, pp. 453–459, Oct. 2021.

[10] M. L. Pinedo, *Scheduling*, vol. 29. New York, NY, USA: Springer, 2012.

[11] J. Xu and X. Zhou, "A class of multi-objective expected value decision-making model with birandom coefficients and its application to flow shop scheduling problem," *Inf. Sci.*, vol. 179, no. 17, pp. 2997–3017, Aug. 2009.

[12] P. Fattahi, S. M. H. Hosseini, and F. Jolai, "A mathematical model and extension algorithm for assembly flexible flow shop scheduling problem," *Int. J. Adv. Manuf. Technol.*, vol. 65, nos. 5–8, pp. 787–802, Mar. 2013.

[13] A. Allahverdi and H. Aydilek, "The two stage assembly flowshop scheduling problem to minimize total tardiness," *J. Intell. Manuf.*, vol. 26, no. 2, pp. 225–237, Apr. 2015.

[14] A. M. Fathollahi-Fard, L. Woodward, and O. Akhrif, "Sustainable distributed permutation flow-shop scheduling model based on a triple bottom line concept," *J. Ind. Inf. Integr.*, vol. 24, Dec. 2021, Art. no. 100233.

[15] S. M. Johnson, "Optimal two- and three-stage production schedules with setup times included," *Nav. Res. Logistics Quart.*, vol. 1, no. 1, pp. 61–68, Mar. 1954.

[16] E. F. Stafford Jr., F. T. Tseng, and J. N. D. Gupta, "Comparative evaluation of MILP flowshop models," *J. Oper. Res. Soc.*, vol. 56, no. 1, pp. 88–101, Jan. 2005.

[17] F. T. Tseng and E. F. Stafford, "New MILP models for the permutation flowshop problem," *J. Oper. Res. Soc.*, vol. 59, no. 10, pp. 1373–1386, Oct. 2008.

[18] C. Gicquel, L. Hege, M. Minoux, and W. van Canneyt, "A discrete time exact solution approach for a complex hybrid flow-shop scheduling problem with limited-wait constraints," *Comput. Oper. Res.*, vol. 39, no. 3, pp. 629–636, Mar. 2012.

[19] C.-X. Wu, M.-H. Liao, M. Karatas, S.-Y. Chen, and Y.-J. Zheng, "Real-time neural network scheduling of emergency medical mask production during COVID-19," *Appl. Soft Comput.*, vol. 97, Dec. 2020, Art. no. 106790.

[20] M. Nawaz, E. E. Enscore, and I. Ham, "A heuristic algorithm for the *m*-machine, *n*-job flow-shop sequencing problem," *Omega*, vol. 11, no. 1, pp. 91–95, Jan. 1983.

[21] S. M. A. Suliman, "A two-phase heuristic approach to the permutation flow-shop scheduling problem," *Int. J. Prod. Econ.*, vol. 64, nos. 1–3, pp. 143–152, Mar. 2000.

[22] Y. Zheng and J. Xue, "A problem reduction based approach to discrete optimization algorithm design," *Computing*, vol. 88, nos. 1–2, pp. 31–54, Jun. 2010.

[23] J. Gao, R. Chen, and W. Deng, "An efficient tabu search algorithm for the distributed permutation flowshop scheduling problem," *Int. J. Prod. Res.*, vol. 51, no. 3, pp. 641–651, Feb. 2013.

[24] W. Shao, D. Pi, and Z. Shao, "Optimization of makespan for the distributed no-wait flow shop scheduling problem with iterated greedy algorithms," *Knowl.-Based Syst.*, vol. 137, pp. 163–181, Dec. 2017.

[25] H. Wei, S. Li, H. Jiang, J. Hu, and J. Hu, "Hybrid genetic simulated annealing algorithm for improved flow shop scheduling with makespan criterion," *Appl. Sci.*, vol. 8, no. 12, p. 2621, Dec. 2018.

[26] O. Engin and A. Güçlü, "A new hybrid ant colony optimization algorithm for solving the no-wait flow shop scheduling problems," *Appl. Soft Comput.*, vol. 72, pp. 166–176, Nov. 2018.

[27] C. Yu, Q. Semeraro, and A. Matta, "A genetic algorithm for the hybrid flow shop scheduling with unrelated machines and machine eligibility," *Comput. Oper. Res.*, vol. 100, pp. 211–229, Dec. 2018.

[28] L. Abualigah, A. Diabat, S. Mirjalili, M. A. Elaziz, and A. H. Gandomi, "The arithmetic optimization algorithm," *Comput. Methods Appl. Mech. Eng.*, vol. 376, Apr. 2021, Art. no. 113609.

[29] S. Katoch, S. S. Chauhan, and V. Kumar, "A review on genetic algorithm: Past, present, and future," *Multimedia Tools Appl.*, vol. 80, no. 5, pp. 8091–8126, Feb. 2021.

[30] K. Keskin and O. Engin, "A hybrid genetic local and global search algorithm for solving no-wait flow shop problem with bi criteria," *Social Netw. Appl. Sci.*, vol. 3, no. 6, pp. 1–15, Jun. 2021.

[31] H. Piroozfard, K. Y. Wong, and A. Hassan, "A hybrid genetic algorithm with a knowledge-based operator for solving the job shop scheduling problems," *J. Optim.*, vol. 2016, Apr. 2016, Art. no. 7319036.

[32] J. Meena, M. Kumar, and M. Vardhan, "Cost effective genetic algorithm for workflow scheduling in cloud under deadline constraint," *IEEE Access*, vol. 4, pp. 5065–5082, 2016.

[33] K. Peng, J. Du, F. Lu, Q. Sun, Y. Dong, P. Zhou, and M. Hu, "A hybrid genetic algorithm on routing and scheduling for vehicle-assisted multi-drone parcel delivery," *IEEE Access*, vol. 7, pp. 49191–49200, 2019.

[34] Y. Wang and Q. Zhu, "A hybrid genetic algorithm for flexible job shop scheduling problem with sequence-dependent setup times and job lag times," *IEEE Access*, vol. 9, pp. 104864–104873, 2021.

[35] M. Amirghasemi and R. Zamani, "An effective evolutionary hybrid for solving the permutation flowshop scheduling problem," *Evol. Comput.*, vol. 25, no. 1, pp. 87–111, Mar. 2017.

[36] M. S. Umam, M. Mustafid, and S. Suryono, "A hybrid genetic algorithm and tabu search for minimizing makespan in flow shop scheduling problem," *J. King Saud Univ.-Comput. Inf. Sci.*, vol. 34, no. 9, pp. 7459–7467, Oct. 2022.

[37] D. Ouelhadj and S. Petrovic, "A survey of dynamic scheduling in manufacturing systems," *J. Scheduling*, vol. 12, no. 4, pp. 417–431, Aug. 2009.

[38] O. Cardin, D. Trentesaux, A. Thomas, P. Castagna, T. Berger, and H. B. El-Haouzi, "Coupling predictive scheduling and reactive control in manufacturing hybrid control architectures: State of the art and future challenges," *J. Intell. Manuf.*, vol. 28, no. 7, pp. 1503–1517, Oct. 2017.

[39] R. Larsen and M. Pranzo, "A framework for dynamic rescheduling problems," *Int. J. Prod. Res.*, vol. 57, no. 1, pp. 16–33, Jan. 2019.

[40] I. Sabuncuoglu and S. Goren, "Hedging production schedules against uncertainty in manufacturing environment with a review of robustness and stability research," *Int. J. Comput. Integr. Manuf.*, vol. 22, no. 2, pp. 138–157, Feb. 2009.

[41] Y.-H. Dong and J. Jang, "Production rescheduling for machine breakdown at a job shop," *Int. J. Prod. Res.*, vol. 50, no. 10, pp. 2681–2691, May 2012.

[42] N. Kundakcı and O. Kulak, "Hybrid genetic algorithms for minimizing makespan in dynamic job shop scheduling problem," *Comput. Ind. Eng.*, vol. 96, pp. 31–51, Jun. 2016.

[43] J. Zhang, W. Qin, L. H. Wu, and W. B. Zhai, "Fuzzy neural network-based rescheduling decision mechanism for semiconductor manufacturing," *Comput. Ind.*, vol. 65, no. 8, pp. 1115–1125, Oct. 2014.

[44] M. Nazari, A. Oroojlooy, L. Snyder, and M. Takác, "Reinforcement learning for solving the vehicle routing problem," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 31, 2018, pp. 1–11.

[45] Y. Li, S. Carabelli, E. Fadda, D. Manerba, R. Tadei, and O. Terzo, "Machine learning and optimization for production rescheduling in Industry 4.0," *Int. J. Adv. Manuf. Technol.*, vol. 110, nos. 9–10, pp. 2445–2463, Oct. 2020.

[46] M. R. Garey, D. S. Johnson, and R. Sethi, "The complexity of flowshop and jobshop scheduling," *Math. Oper. Res.*, vol. 1, no. 2, pp. 117–129, May 1976.

[47] O. Etiler, B. Toklu, M. Atak, and J. Wilson, "A genetic algorithm for flow shop scheduling problems," *J. Oper. Res. Soc.*, vol. 55, no. 8, pp. 830–835, 2004.

[48] A. Konak, D. W. Coit, and A. E. Smith, "Multi-objective optimization using genetic algorithms: A tutorial," *Rel. Eng. Syst. Saf.*, vol. 91, no. 9, pp. 992–1007, Sep. 2006.

[49] C. W. Ahn and R. S. Ramakrishna, "A genetic algorithm for shortest path routing problem and the sizing of populations," *IEEE Trans. Evol. Comput.*, vol. 6, no. 6, pp. 566–579, Dec. 2002.

[50] M. Srinivas and L. M. Patnaik, "Adaptive probabilities of crossover and mutation in genetic algorithms," *IEEE Trans. Syst., Man, Cybern.*, vol. 24, no. 4, pp. 656–667, Apr. 1994.

[51] M. Chen, J. Wen, Y.-J. Song, L.-N. Xing, and Y.-W. Chen, "A population perturbation and elimination strategy based genetic algorithm for multi-satellite TT&C scheduling problem," *Swarm Evol. Comput.*, vol. 65, Aug. 2021, Art. no. 100912.

[52] Y. Du, T. Wang, B. Xin, L. Wang, Y. Chen, and L. Xing, "A data-driven parallel scheduling approach for multiple agile earth observation satellites," *IEEE Trans. Evol. Comput.*, vol. 24, no. 4, pp. 679–693, Aug. 2020.

[53] M. Squires, X. Tao, S. Elangovan, R. Gururajan, X. Zhou, and U. R. Acharya, "A novel genetic algorithm based system for the scheduling of medical treatments," *Expert Syst. Appl.*, vol. 195, Jun. 2022, Art. no. 116464.

[54] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014, *arXiv:1412.6980*.

[55] L.-Y. Tseng and Y.-T. Lin, "A hybrid genetic algorithm for no-wait flowshop scheduling problem," *Int. J. Prod. Econ.*, vol. 128, no. 1, pp. 144–152, Nov. 2010.

[56] J.-Y. Ding, S. Song, J. N. D. Gupta, R. Zhang, R. Chiong, and C. Wu, "An improved iterated greedy algorithm with a tabu-based reconstruction strategy for the no-wait flowshop scheduling problem," *Appl. Soft Comput.*, vol. 30, pp. 604–613, May 2015.

[57] F. Zhao, H. Liu, Y. Zhang, W. Ma, and C. Zhang, "A discrete water wave optimization algorithm for no-wait flow shop scheduling problem," *Expert Syst. Appl.*, vol. 91, pp. 347–363, Jan. 2018.

[58] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: MIT Press, 2018.

PAKKAPORN SAOPHAN received the B.S. degree in management mathematics from Thammasat University, Thailand, in 2017, and the M.S. degree in management mathematics from the Sirindhorn International Institute of Technology (SIIT), Thammasat University, Thailand, in 2018. She is currently pursuing the Ph.D. degree in knowledge science with the Japan Advanced Institute of Science and Technology (JAIST), Japan. Her research interest includes developing and incorporating machine learning for Industry 4.0 manufacturing.

WARUT PANNAKKONG received the B.Engr. degree in industrial engineering and the M.Engr. degree in logistics and supply chain systems engineering from the Sirindhorn International Institute of Technology (SIIT), Thammasat University, Thailand, in 2010 and 2014, respectively, and the Ph.D. degree in knowledge science from the Japan Advanced Institute of Science and Technology (JAIST), in 2017. He is currently an Associate Professor with the School of Manufacturing Systems and Mechanical Engineering (MSME), SIIT, Thammasat University. His research interests include artificial intelligence for industry, time series forecasting, data mining, machine learning, computer vision in industrial applications, discrete event system simulation and optimization, production planning, and logistics and supply chain management.

RAVEEKIAT SINGHAPHANDU received the B.S. degree in computer science from the Sirindhorn International Institute of Technology (SIIT), Thammasat University, Thailand, in 2014, and the M.S. degree in informatics from the Technical University of Munich, Germany, in 2017. He is currently pursuing the Ph.D. degree with SIIT, Thammasat University, and the Japan Advanced Institute of Science and Technology, Japan. His research interests include machine learning, computer vision, and software engineering in industrial engineering applications.

VAN-NAM HUYNH (Member, IEEE) received the Ph.D. degree in mathematics from the Vietnam Academy of Science and Technology, in 1999. He is currently a Professor with the School of Knowledge Science, Japan Advanced Institute of Science and Technology (JAIST). His current research interests include machine learning and data mining, AI reasoning, argumentation, multi-agent systems, decision analysis, management science, and Kansei information processing and applications. He currently serves as an Area Editor for *International Journal of Approximate Reasoning*, the Editor-in-Chief for *International Journal of Knowledge and Systems Science*, and the Editorial Board Member of the *Array* journal.