electronics

Article

Rescheduling of Distributed Manufacturing System with Machine Breakdowns

Xiaohui Zhang [1], Yuyan Han [2], Grzegorz Królczyk [3], Marek Rydel [4], Rafal Stanislawski [4] and Zhixiong Li [3,*]

[1] School of Electrical and Control Engineering, Xuzhou University of Technology, Xuzhou 221018, China; xh_zhang@xzit.edu.cn
[2] School of Computer Science, Liaocheng University, Liaocheng 252000, China; hanyuyan@lcu-cs.com
[3] Department of Manufacturing Engineering and Automation Products, Opole University of Technology, 45-758 Opole, Poland; g.krolczyk@po.opole.pl
[4] Department of Electrical, Control and Computer Engineering, Opole University of Technology, 45-758 Opole, Poland; m.rydel@po.opole.pl (M.R.); r.stanislawski@po.edu.pl (R.S.)
* Correspondence: z.li@po.edu.pl

Abstract: This study attempts to explore the dynamic scheduling problem from the perspective of operational research optimization. The goal is to propose a rescheduling framework for solving distributed manufacturing systems that consider random machine breakdowns as the production disruption. We establish a mathematical model that can better describe the scheduling of the distributed blocking flowshop. To realize the dynamic scheduling, we adopt an "event-driven" policy and propose a two-stage "predictive-reactive" method consisting of two steps: initial solution pregeneration and rescheduling. In the first stage, a global initial schedule is generated and considers only the deterministic problem, i.e., optimizing the maximum completion time of static distributed blocking flowshop scheduling problems. In the second stage, that is, after the breakdown occurs, the rescheduling mechanism is triggered to seek a new schedule so that both maximum completion time and the stability measure of the system can be optimized. At the breakdown node, the operations of each job are classified and a hybrid rescheduling strategy consisting of "right-shift repair + local reorder" is performed. For local reorder, we designed a discrete memetic algorithm, which embeds the differential evolution concept in its search framework. To test the effectiveness of DMA, comparisons with mainstream algorithms are conducted on instances with different scales. The statistical results show that the ARPDs obtained from DMA are improved by 88%.

Keywords: distributed manufacturing; rescheduling; memetic algorithm

check for updates

Citation: Zhang, X.; Han, Y.; Królczyk, G.; Rydel, M.; Stanislawski, R.; Li, Z. Rescheduling of Distributed Manufacturing System with Machine Breakdowns. *Electronics* **2022**, *11*, 249. https://doi.org/10.3390/electronics11020249

Academic Editors: Darius Andriukaitis and Peter Brida

Received: 5 December 2021
Accepted: 10 January 2022
Published: 13 January 2022

1. Introduction

With the advancement of economic globalization and the intensification of mergers between enterprises, the emergence of large-scale or concurrent production makes the pattern of distributed manufacturing necessary [1,2]. Distributed manufacturing decentralizes tasks into factories or workshops from different geographical locations. This pattern can help the manufacturers raise productivity, reduce cost, control risks, and adjust marketing policies more flexibly [3]. As an important part of distributed manufacturing, scheduling directly affects the efficiency and competitiveness of enterprises. Generally speaking, to solve such problems, a problem-specific model with production constraints should be first established to describe the scheduling problem considered. Then, optimization methods (e.g., mathematical programming, intelligent optimization, etc.) of operational research are developed to search for an optimal solution. For systems with large-scale and high complexity, mathematical programming such as integer programming, branch and bound, dynamic programming, or cut plane can rarely find an optimal solution (ranking) in the target space due to enumeration concept, but the efficiency decreases with the increment of the number of jobs/tasks to be scheduled.

At present, most studies use intelligent optimization algorithms to approximate the optimal solution for scheduling problems. The intelligent optimization algorithm, also called the evolutionary optimization algorithm, or metaheuristic, reveals the design principle of optimization algorithm through the understanding of relevant behavior, function, rules, and action mechanism in biological, physical, chemical, social, artistic, and other systems or fields. It refines the corresponding feature model under the guidance of the characteristics of specific problems and designs an intelligent iterative search process. That is, these kinds of algorithms do not rely on the characteristics of problems, but obtain near-optimal solutions through continuous iterations of global and local search. When an intelligent algorithm is applied for scheduling problems, it can express the schedule as a permutation model in the form of coding, and further compress the solution space into a very flat space, so that a large number of different permutations (schedules) correspond to the same target. Hence, the permutation model-based algorithm can search more different schedules in the target space range in tens of milliseconds to tens of seconds, so as to obtain a solution better than the traditional mathematical programming method.

The object of this study is related to the distributed blocking flowshop scheduling problem (DBFSP) [4]. Figure 1 illustrates DBFSP, which considers f parallel factories that contain the same machine configurations and technological processes [5]. The jobs can be assigned to any factories and each job follows the same blocking manufacturing procedure [6]. Although the machines configured in each distributed factory are the same, the processing time of each operation of each job is assumed to be different, thereby the processing tasks assigned to each distributed factory and their completion time are also different. The idea of solving DBFSP is to reasonably allocate the jobs to the factory through optimization algorithms, and then sequence the jobs in each distributed factory, to optimize the manufacturing objectives of the whole work order. Currently, researchers have made great efforts on solving DBFSP in a static environment, the existing researches mainly focused on the construction of mathematical models and the design of optimization algorithms. Zhang et al. [7] have established two different mathematical models using forward and reverse recursion approaches. A hybrid discrete differential evolution (DDE) algorithm was proposed to minimize the maximum completion time (makespan). Zhang et al. [8] constructed the mixed-integer model for DBFSP and developed a discrete fruit fly algorithm (DFOA) with a speed-up mechanism to minimize the global makespan. Additionally, Shao et al. [9] proposed a hybrid enhanced discrete fruit fly optimization algorithm (HEDFOA) to optimize the makespan. A new assignment rule and an insertion-based improvement procedure were developed to initialize the common central location of different fruit fly swarms. Li et al. [10] investigated a special case of DBFSP, in which a transport robot was embedded in each factory. The loading and unloading times are considered and different for all of the jobs conducted by the robot. An improved iterated greedy (IIG) algorithm was proposed to improve productivity. Moreover, Zhao et al. [11] proposed an ensemble discrete differential evolution (EDE) algorithm, in which three initialization heuristics that consider the front delay, blocking time, and idle time were designed. The mutation, crossover, and selection operators are redesigned to assist the EDE algorithm to execute in the discrete domain.

The above researches on DBFSP have formed a certain system, but they assumed that no explicit disruptions occur during the manufacturing process. In fact, a series of uncertainties often happened during the manufacturing process [12]. These uncertainties, which are sudden and uncontrollable, can change the state of the system strongly and affect the scheduling activities continuously [13]. As a result, the original static schedules are no longer suitable for real-time scheduling. To eliminate the impact of sudden uncertainties, rescheduling operations are generally performed in response to disruptions [14,15]. Rescheduling refers to the procedure of modifying the existing schedule to obtain a new feasible one after uncertain events occur [16]. One of the most important rescheduling strategies for traditional flowshop is "predictive-reactive" scheduling [17]. "predictive-reactive" scheduling defines a two-stage "event-driven" scheduling operation: the first stage generates an initial schedule that provides a baseline reference for other manufactural activities such as procurement and distribution of raw materials [18]. Influenced by the disruptions, the second stage explicitly quantifies the disruptions, constructs the management model with the disruption information gathered by the cyber-physical smart manufacturing technology [19–21], adjusts the initial schedule, and makes an effective trade-off between the initial optimization objective and the disturbance objective [22]. Since little literature is on the rescheduling of DBFSP, we review only the rescheduling strategies and algorithms developed for traditional and single flowshop. To realize the rescheduling, a suitable strategy should be determined in advance according to the scenario. Framinan et al. [23] discussed the problem of high system tension caused by continuous rescheduling of multi-stage flow production. A rescheduling strategy was described by estimating the availability of the machines after disruptions and a reordering algorithm based on the critical path was proposed. Katragjini et al. [24] analyzed eight types of uncertainties and designed rescheduling strategies through the classification of job status, which considers the completed, in processed and unprocessed operations. Iris et al. [25] designed a recoverable strategy taking the uncertainty of crane arrival to the ship and the fluctuation of loading and unloading speeds into account. The rescheduling strategy used a proactive baseline with reactive costs as the objective. Ma et al. [26] took the overmatch time (difference between real manufacturing time and the estimated time of the initial schedule) as one of the objectives in the rescheduling model to handle production emergencies in parallel flowshops. Li et al. [27] discussed both machine breakdown and processing change interruptions for a hybrid flowshop. The authors have proposed a hybrid fruit fly optimization algorithm (HFOA) with processing-delay, cast-break erasing, and right-shift strategy to minimize different rescheduling objectives in a steelmaking-foundry system. Li et al. [28] also considered five types of interruption events in the flowshop, namely machine breakdown, new jobs arrival, jobs cancellation, job processing change, and job release time change. A rescheduling strategy based on job status was designed for each event. A discrete teaching and learning optimization (DTLO) algorithm was proposed to optimize the makespan and stability. Valledor et al. [29] applied the Pareto optimum to solve the multi-objective flowshop rescheduling problem with makespan, total weighted tardiness, and steadiness as objectives. Three classes of disruptions (appearance of new jobs, machine faults, and changes in operational times) were discussed and an event management model was constructed. A restarted iterated Pareto greedy (RIPG) metaheuristic is used to find the optimal Pareto front.

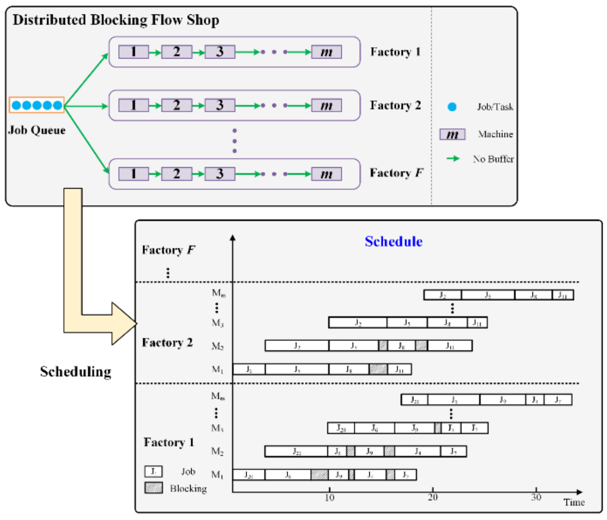

Figure 1. An example of DBFSP with the Gantt chart.

From the above review, it can be concluded that current researches focused mostly on the rescheduling of a single flowshop with various constraints. Little literature has considered rescheduling from the distributed manufacturing perspective. Though the Industry 4.0 wireless networks [30,31] have quickly developed in recent years, they are involved more in distributed information interconnection rather than decision making in scheduling fields. Likewise, the big data-driven technology [32,33] may provide real-time decisions or schedule rules for small-scale manufacturing, but has not formed a sound system. Moreover, big data technology relies strongly on a large amount of historical data, it is difficult to apply to new products due to the highly discrete, stochastic, and distributed properties of scheduling problems. Therefore, with the in-depth application of distributed manufacturing, distributed rescheduling strategies and approaches need to be formulated prudently so that effective references could be provided for modern decision-makers.

On the other hand, the objects of job shop scheduling are usually individual jobs, products, or other resources in the manufacturing process. Such resources have typical discrete characteristics, which need to be marked and expressed through special information carriers, and then obtain new combinations (ranking) by constantly updating the information carriers. These optimization characteristics are similar to the optimization process of the intelligent algorithm based on the permutation model. Therefore, the intelligent algorithm based on the evolution concept is more suitable for solving scheduling problems.

According to the above analysis and good applicability of the intelligent algorithm, we use an intelligent algorithm to reschedule the distributed blocking flowshop scheduling problem in a dynamic environment (DDBFSP). In the last decade, the application of intelligent algorithms for solving scheduling problems has been extensively investigated. Memetic algorithm (MA), also called the Lamarckian evolutionary algorithm, is attracting increasing concern. The concept of "meme" refers to contagious information patterns proposed by Dawkins in 1976 [34]. "Memes" are similar to genes in GA, but there are differences: Memetic evolution is characterized by Lamarckism, while genetic evolution is characterized by Darwinism. Meanwhile, neural system-based memetic information is more malleable than genetic information, so memes are more likely to change and spread more quickly than genes. In evolutionary computing, MA can combine various global and local strategies to construct different search frameworks, which possess the characteristics of GA but with stronger merit-seeking ability. MA was widely applied in many engineering problems, such as vehicle path planning [35], home care routing [36], bin packing problem [37], broadcast resource allocation [38], and production scheduling optimization [39–41]. Until now, MA has not been applied to solve DBFSP in a dynamic environment(DDBFSP), it will be of significance to extend MA as a solver for DDBFSP.

In summary, this paper aims to optimize DDBFSP with both makespan and stability measures as the objectives. The machine breakdown is defined as the disruption and assumed to happen stochastically in any distributed factories. To handle such dynamic events, a problem-specific disruption management model is constructed. A rescheduling framework that includes a job status-oriented classification strategy and a reordering algorithm-discrete Memetic algorithm (DMA) is proposed. For DMA, the differential evolution (DE) operators have been embedded to execute the neighborhood search. A simulated annealing (SA)-based reference local search framework is designed to help the algorithm escape from local optimums. Finally, the effectiveness of DMA is validated through comparative experiments. It is expected that the effect after rescheduling is to highly maintain the level of optimization of the original manufacturing objective (makespan) while ensuring the stability of the newly generated schedules.

The remainder of the paper is organized as follows. Section 2 states DDBFSP and constructs the mathematical model and objective function for DDBFSP. Section 3 designs the corresponding rescheduling framework. Section 4 elaborates the details of the DMA reordering algorithm. Section 5 verifies the performance of DMA and analyzes the results; Section 6 summarizes the research content of this paper.

2. Method

In this section, the mathematical models for DBFSP with optimization objectives in both static and dynamic environments are proposed. The classifications of job status after breakdown events are also introduced.

2.1. Statement of DBFSP in Static Environment

As can be seen in Figure 1, DBFSP not only needs to consider the correlation between processing task characteristics and blocking constraints but also needs to consider the coupling of global scheduling and local scheduling of each distributed factory, the solving process is more complex. As illustrated in Figure 1, a set of jobs $J = \left\{J_j|1,2,\ldots,n\right\}$ will be assigned to a set of factories $F = \{F_k|k=1,2,\ldots,f\}$, each of which contains a set of machines $M = \{M_i|1,2,\ldots,m\}$. The blocking constraint determines that no buffers are allowed between two adjacent machines. Therefore, the job can only be released to the next operation when the subsequent machine is free; otherwise, the job must be blocked on the current machine. We assume the processing time for each job is stochastic and different. After a job is assigned to a processing plant, it is not allowed to move to other factories.

Assume $n_k(n_k \leq n)$ jobs are assigned to factory k, and the job sequence in this factory is denoted as π_k, where $\pi_k(l)$ represents the l-th job in π_k. The operation $O_{\pi_k(l),i}$ has a processing time $P_{\pi_k(l),i}$. Assume $S_{\pi_k(l),0}$ is the start time of $\pi_k(l)$ on the first machine of factory k, and $d_{\pi_k(l),i}$ is defined as the departure time of operation $O_{\pi_k(l),i}$ on machine i. The recursive formulas of DBFSP can be derived as follows:

$$S_{\pi_k(1),0} = 0 \tag{1}$$

$$D_{\pi_k(1),i} = D_{\pi_k(1),i-1} + P_{\pi_k(1),i}, \quad i = 2,3,\ldots,m \tag{2}$$

$$S_{\pi_k(l),0} = D_{\pi_k(l-1),1}, \;\; i = 2,\ldots,n_k \tag{3}$$

$$D_{\pi_k(l),i} = \max\left\{D_{\pi_k(l),i-1} + p_{\pi_k(l),i}, \;\; D_{\pi_k(l-1),i+1}\right\}, \quad l = 2,3,\ldots,n_k \quad i = 1,2,\ldots,m-1 \tag{4}$$

$$D_{\pi_k(l),m} = D_{\pi_k(l),m-1} + P_{\pi_k(l),m}, \quad l = 2,3,\ldots,n_k \tag{5}$$

In the above equations, Equations (1) and (2) calculate the start and departure time of the first job π_k from machine 1 to the last machine m in factory k. Equations (3) and (4) calculate the start and departure time of job $\pi_k(l)$ from machine 1 to machine $m-1$. Equation (5) gives the departure time of $\pi_k(l)$ on machine m. If we take makespan $C(\pi_k)$ as the optimization objective, $C(\pi_k)$ of factory k can be expressed as:

$$C(\pi_k) = D_{\pi_k(n_k),m} \tag{6}$$

As a result, the global makespan for DBFSP is defined through the comparison between $C(\pi_k)$ of all distributed factories:

$$C_{\max}(\Pi) = \max_{k=1}^{f}(C(\pi_k)) \tag{7}$$

The detailed recursive process and example refer to our previous group work [8] for solving DBFSP.

2.2. Statement of DDBFSP

When characterizing the machine breakdown event of DBFSP in a dynamic and stochastic environment, the following questions should be marked: (1) Which factory has happened the breakdown event and when (the probability of breakdown)? (2) Which machine in that factory breaks (the distributivity of the breakdown)? (3) When will the machine resume?

In fact, machine breakdowns are difficult to simulate since the probabilistic model of breakdown occurrence could hardly cover the real manufacturing situation. Moreover, the recovery time is mostly predicted based on a priori knowledge, which cannot guarantee accuracy. With consideration of randomness and distribution, this paper triggers machine breakdowns in a randomly selected distributed factory at time t. The breakdown time is defined to follow a discrete uniform distribution function which is expressed as follows:

$$E(B_{k,i}) = rand()\%P(T_i) + p_{\pi_k(l)} \times i, \quad i = 1,2,3\ldots,m,\ k = 1,2,\ldots,f,\ l = 1,2,\ldots,n_k \tag{8}$$

where $rand()$ represents the random function between $[0, \omega]$ and ω denotes the maximum constant of the system; "%" is the remainder operator; $P(T_i)$ represents the total processing time of all jobs on machine i. Equation (8) defines the time range of the breakdown in the factory k during the processing time of all jobs, i.e., $[P_{\pi_k(l),i}, C_{\pi_k(l),i}]$.

To maintain the distribution of breakdowns and the convenience of experimental statistics, this study assumes that each distributed factory occurs β times random breakdowns during manufacturing. Additionally, to maintain the randomness of breakdown occurrence, each machine has the same probability to break down. To ensure a stochastic dynamic environment, the duration of breakdowns are generated randomly and uniformly following the interval $[0, \omega]$ and are determined immediately after the event. Moreover, other constraints for the breakdown event in DDBFSP are defined as follows:

(1) All machines exist three statuses during manufacturing: idle, processing and blocked, a breakdown event occurs in the processing period.
(2) The system triggers one machine breakdown each time, and the process stops immediately when the breakdown occurs.
(3) After the machine is recovered, the affected process can continue with the remaining processing without re-processing.

2.3. Optimization Objectives of DDBFSP

In DBFSP, it is generally necessary to consider the production efficiency-related objective, e.g., makespan. While in a dynamic environment, from the decision point of view, stability becomes one importantly practical metric for manufacturing systems. If rescheduling optimization is performed only considering the production efficiency-related indicators, it may generate new schedules that deviate significantly from the initial plan, which in turn affects other planning activities, such as material management and manpower planning. Therefore, in the rescheduling phase, in addition to makespan, the stability of the new schedules should be considered. In this study, the initial schedule of one distributed factory before a breakdown occurs is denoted by B (Baseline), and the schedule after rescheduling is denoted by $B*$. The goal of rescheduling is to optimize both the makespan and the stability of the distributed factory at each breakdown node. The first objective (makespan) of the DDBFSP is expressed as follows:

$$f_1 = C_{\max}(B*) \tag{9}$$

The second objective (stability) of the DDBFSP is derived as follows.

$$f_2 = \min\{\sum_{i=1}^{m}\sum_{l=1}^{n_k} Z_{\pi_k(l),i}\}, \quad k = 1,2,\ldots,f,\ n_k = 1,2,\ldots,n \tag{10}$$

where the decision variable $Z_{\pi_k(l),i}$ indicates whether the relative position of the job B and $B*$ has changed. $Z_{\pi_k(l),i} = 1$ represents that the position of job $\pi_k(l)$ on machine i in factory k has changed in the new schedule $B*$ and vice versa $Z_{\pi_k(l),i} = 0$.

To simplify the optimization process and avoid redundant calculations, a weighting mechanism is applied to combine both objective functions:

$$f(B*) = w_1 * f_1 + w_2 * f_2 \tag{11}$$

In Equation (11), w_1 and w_2 represent the weight coefficients of f_1 and f_2, respectively. Since f_1 and f_2 have different distribution ranges of dimensions, to avoid the results being dominated by the data with larger or smaller distribution ranges, the normalization method proposed in [24] is applied so that the value range of each objective falls in the interval. The normalization function is defined as follows:

$$f(B*) = w_1 * N(f_1) + w_2 * N(f_2) \tag{12}$$

where:

$$N(f_1) = \frac{f_1(B*) - low(f_1)}{up(f_1) - low(f_1)} \tag{13}$$

$$N(f_2) = \frac{f_2(B*) - low(f_2)}{up(f_2) - low(f_2)} \tag{14}$$

In Equations (13) and (14), $up(.)$ and $low(.)$ represent the upper and lower bounds of f_1 and f_2 for the two extreme rankings of the jobs at the breakdown node, respectively. The specific calculation procedure refers to [24].

2.4. Statement of Job Status after Breakdown Event

After a machine breaks down, the jobs are categorized to construct the event management model:

$$C^*_{\pi_k(l),i} - S^*_{\pi_k(l),i} - P^*_{\pi_k(l),i} + (1 - y_{\pi_k(l),i,1})\omega \geq 0 \tag{15}$$

$$C^*_{\pi_k(l),i} - S^*_{\pi_k(l),i} - P^*_{\pi_k(l),i} - (1 - y_{\pi_k(l),i,1})\omega \leq 0 \tag{16}$$

$$C^*_{\pi_k(l),i} - S^*_{\pi_k(l),i} - P^*_{\pi_k(l),i} - B_{e,i} + B_{s,i} + (1 - y_{\pi_k(l),i,2})\omega \geq 0 \tag{17}$$

$$C^*_{\pi_k(l),i} - S^*_{\pi_k(l),i} - P^*_{\pi_k(l),i} - B_{e,i} + B_{s,i} - (1 - y_{\pi_k(l),i,2})\omega \leq 0 \tag{18}$$

$$C^*_{\pi_k(l),i} - \max\left\{S^*_{\pi_k(l),i}, B_{e,i}\right\} - P^*_{\pi_k(l),i} + (1 - y_{\pi_k(l),i,3})\omega \geq 0 \tag{19}$$

$$C^*_{\pi_k(l),i} - \max\left\{S^*_{\pi_k(l),i}, B_{e,i}\right\} - P^*_{\pi_k(l),i} - (1 - y_{\pi_k(l),i,3})\omega \leq 0 \tag{20}$$

$$\sum_{g=1}^{3} Y_{\pi_k(l),i,g} = 1, \quad i = \{1, 2, \ldots, m\},\ k = \{1, 2, \ldots, f\},\ g = \{1, 2, 3\} \tag{21}$$

$$Y_{\pi_k(l),i,g} = \{0, 1\}, \quad i = \{1, 2, \ldots, m\},\ k = \{1, 2, \ldots, f\},\ g = \{1, 2, 3\} \tag{22}$$

In the above equations, $C^*_{\pi_k(l),i}$ denotes the completion time of job $\pi_k(l)$ on the machine i in $B*$. $S^*_{\pi_k(l),i}$ and $P^*_{\pi_k(l),i}$ represent the corresponding start time and processing time of $C^*_{\pi_k(l),i}$, respectively. $B_{e,i}$ and $B_{s,i}$ denote the occurrence time and completion time of the breakdown on the machine i. Equation (15) to Equation (20) defines three statuses in which an operation of a job is in when a breakdown occurs: Equations (15) and (16) determine that the operation was completed before the breakdown occurs; Equations (17) and (18) determine that the operation is being processed when the breakdown occurs; Equations (19) and (20) determine that the operation was originally scheduled to start after the machine is recovered. Equation (21) represents that the job can only be in a state in case a breakdown occurs. Equation (22) is a binary decision variable set for three cases: (1) $Y_{\pi_k(l),i,1} = 1$ means the operation is completed before the breakdown occurs; (2) $Y_{\pi_k(l),i,2} = 1$ denotes the operation overlaps with the machine breakdown node; (3) $Y_{\pi_k(l),i,3} = 1$ means the operation begins after the machine is resumed.

For a better understanding, an example is presented in Figure 2 to illustrate the classification of the operation status at the moment that a machine breaks down. In case 1 of Figure 2, machine 2 of factory k breaks down at time 55, at which time the operations $O_{\pi_k(1),1}$, $O_{\pi_k(1),2}$ and $O_{\pi_k(2),1}$ have already completed processing. Their start and completion times are not affected and do not need to be adjusted in the rescheduling phase. In case 2 of Figure 2, machine 1 breaks down at node 55 and operation $O_{\pi_k(3),1}$ is being processed at this time. The breakdown divides $O_{\pi_k(3),1}$ into two parts: the finished and the remaining part, the remaining part is completed after the machine is recovered. Hence, the start time $O_{\pi_k(3),1}$ remains unchanged in the rescheduling phase, but its finish time is affected by both the breakdown time and the recovery time. In case 3 of Figure 2, machine 1 breaks down at node 38 which is before the startup of job J_3 and J_4. Therefore, the start and finish times of J_3 and J_4 are affected by both the breakdown time and the recovery time.

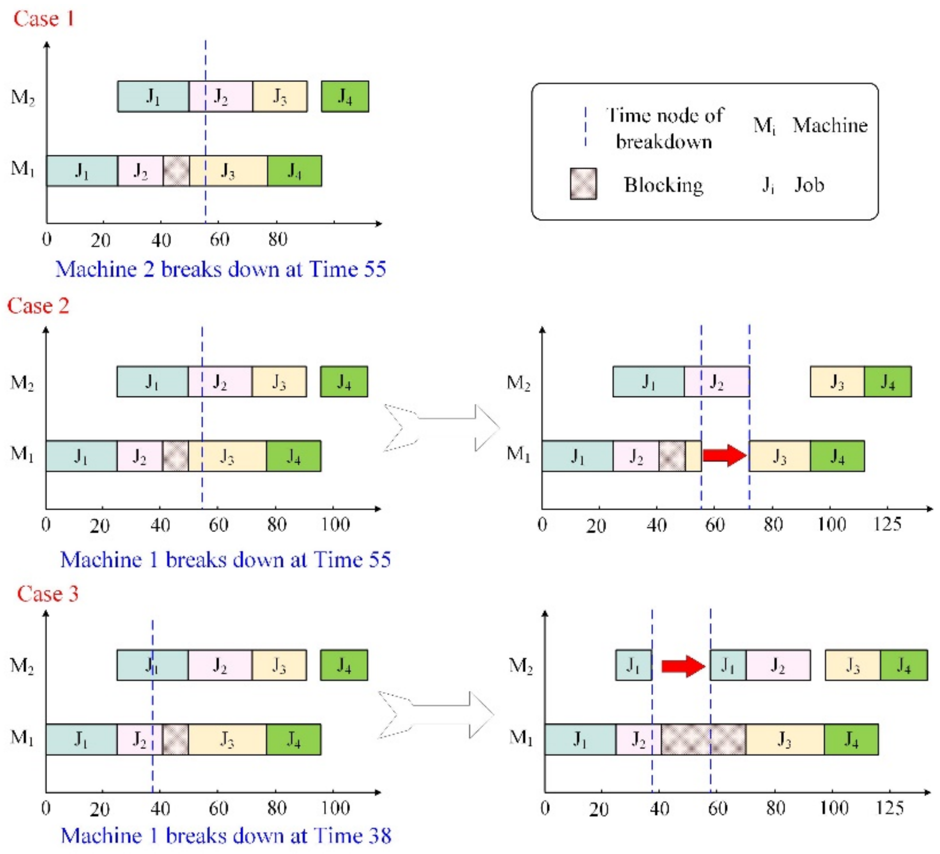

Figure 2. Classification of job status at machine breakdowns.

3. Rescheduling Framework for DDBFSP

Since the breakdowns occur during the manufacturing procedure, on which each job has already been fixed in a certain factory for processing, a change of jobs between different factories is unrealistic. Hence, the individual factory is defined as the decision subject in response to the events.

3.1. Rescheduling Strategy

We envision a stochastic and dynamic distributed scheduling environment. A two-stage "predictive-reactive" method is proposed for DDBFSP: initial solution pre-generation and rescheduling. In the first stage, the initial schedule for DDBFSP is generated by considering a static environment without machine breakdowns. After the breakdown occurs, the initial schedule may no longer be optimal. Therefore, in the second stage, the "event-driven" rescheduling is triggered to evaluate the breakdown, and a new schedule is provided in response to the events. Since the new schedule will be executed until the next breakdown occurs, the rescheduling strategy proposed in this study should have a dual objective: on one hand, to adapt and minimize the impact of the event; on the other hand, to generate a schedule that gives a good tradeoff between scheduling quality and stability when the event is resumed.

3.2. Rescheduling Method

According to the classification of operation status in Section 2.3, we propose a hybrid rescheduling method: "right-shift repair + local reorder". At the breakdown node, no adjustment is made to the completed operations; for the directly affected operations, the right-shift strategy is adopted for a local repair; for the jobs that have not yet started their processing, the reordering algorithm is performed to seek a better partial schedule. The proposed "right-shift repair + local reorder" method is described as follows:

First, determine the jobs that are to be rescheduled. According to the constraint limitation of the flowshop manufacturing, once the processing sequence of the jobs on the first machine is determined, their sequence on other machines must be the same. Therefore, we mark the jobs $\pi_k(l)$ at the breakdown node by using the first machine as the reference. Suppose that there is a breakdown event at time $B_{s,i}$ when job $\pi_k(l)$ is processed on machine i, then the jobs in this factory of which the first operation has been started are counted in the set N_c; relatively, the jobs of which the first operation has not been started are counted in the set of unprocessed jobs N_n.

Subsequently, the jobs N_c are further divided by taking $\pi_k(l)$ as the midpoint: the jobs sequencing before $\pi_k(l)$ are included in the set $N_{c,c} = \{\pi_k(1), \ldots, \pi_k(l-1)\}$; the remaining jobs N_c are included in the set $N_{c,n}$. The rescheduling system maintains the same order and time points for the operations of jobs $N_{c,c}$. For the operations of jobs in $N_{c,n}$, the unaffected operations remain unchanged; the affected and other unprocessed operations are adjusted using the right-shift repair method [42], which shifts the start time to right by certain matching time units. The right-shift repair is essentially a FIFO-based heuristic.

At the breakdown node, all jobs N_n have not been started processing. Their initial order may be no longer optimal after the recovery. Thus, we propose an improved algorithm to reorder the jobs. Eventually, the new schedule is merged into the global schedule and is executed as the baseline until the next breakdown occurs.

To describe the proposed "right-shift repair + local reorder" method more clearly, Figure 3 shows a comparative example with different rescheduling methods at the time of breakdown in one distributed factory. As shown in Figure 3a, the initial schedule of the factory has a desired makespan of 36. During manufacturing, Machine 2 breaks down at time $B_{s,i} = 8$ and is assumed to be recovered at $B_{e,i} = 11$. At this point, the jobs that have already been processed on machine 1 are J_2 and J_1 (marked in gray), and these two jobs are counted in the set N_c. In contrast, jobs J_3, J_4 and J_5 are counted in N_n. The framework adjusts the affected operations in N_c based on the breakdown information and reorders the jobs in N_n. As seen in Figure 3b, the right-shift method is implemented on partial operations of the jobs (marked in yellow), the process order remains the same as $J_2 \rightarrow J_1 \rightarrow J_5 \rightarrow J_3 \rightarrow J_4$ and the makespan is delayed to 41. In Figure 3c while using the "right-shift repair + local reorder" method, job J_3, J_4 and J_5 (marked in green) is reordered using the reordering algorithm. The process order changes to $J_2 \rightarrow J_1 \rightarrow J_3 \rightarrow J_4 \rightarrow J_5$ and the makespan is 37, which absorbs 4 units of the recovery time. This example proves that the proposed rescheduling method is more efficient and flexible than the single rule-based (right-shift repair) heuristics.

3.3. Rescheduling Procedure

We illustrate the proposed optimization procedure for DDBFSP in Figure 4. First, with makespan as the optimization objective, a global schedule for DBFSP in a static environment is generated using DFOA [8]; then, each distributed factory executes manufacturing tasks according to their initial schedules; the breakdowns are triggered following the discrete generation mechanism proposed in Section 2.2. Each time the breakdown happens, rescheduling is implemented by the single distributed factory, with makespan and stability as optimization objectives. According to process status at the breakdown node, the jobs are classified and different methods (right-shift repair or reordering algorithm) are performed. The rescheduling results of different job sets are integrated, and the updated schedule under a single breakdown is provided as the initial schedule before the next event occurs. The above procedure is repeated until the termination condition of the breakdown trigger is met, and the final schedule is output.

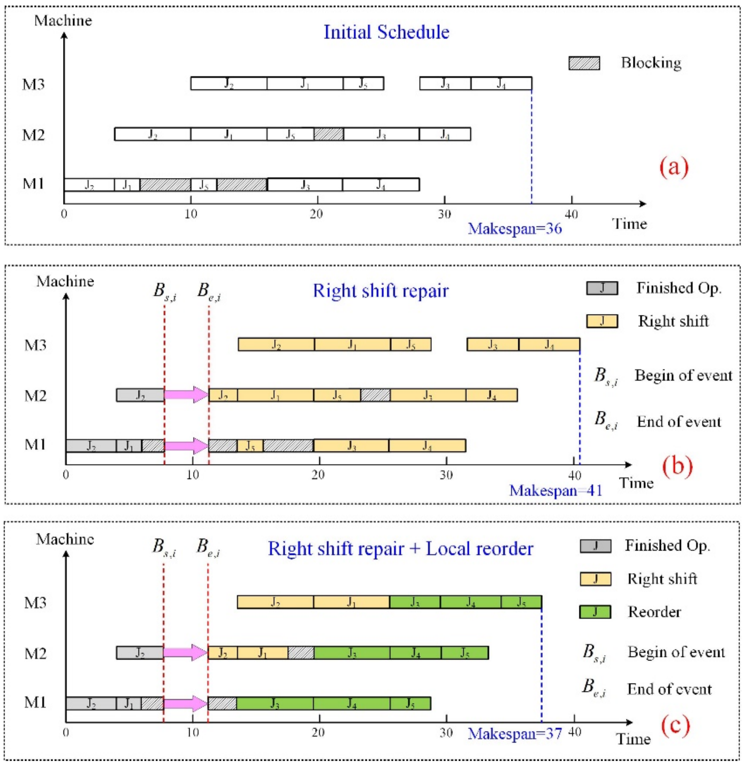

Figure 3. Comparison of different rescheduling strategies.

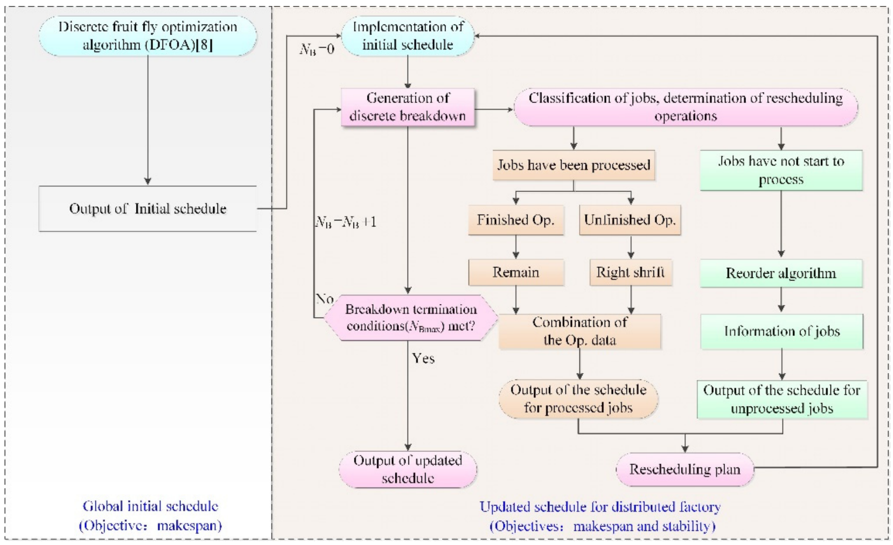
Discrete fruit fly optimization algorithm (DFOA)[8]
Output of Initial schedule
Global initial schedule (Objective: makespan)
Implementation of initial schedule
$N_B=0$
Generation of discrete breakdown
Classification of jobs, determination of rescheduling operations
Jobs have been processed
Jobs have not start to process
Finished Op.
Unfinished Op.
Remain
Right shrift
Reorder algorithm
$N_B=N_B+1$
No
Breakdown termination conditions(N_{Bmax}) met?
Yes
Combination of the Op. data
Information of jobs
Output of the schedule for processed jobs
Output of the schedule for unprocessed jobs
Output of updated schedule
Rescheduling plan
Updated schedule for distributed factory (Objectives: makespan and stability)

Figure 4. Proposed rescheduling procedure for DDBFSP.

4. Reordering Algorithm-DMA

Since the "right-shift repair" is introduced in [42], this section introduces the proposed reordering algorithm-DMA for the jobs in the set N_n.

4.1. Introduction of Standard MA

MA was initially defined as an improvement of GA, it can combine different global and local strategies to construct various search frameworks, which has stronger flexibility, and merit-seeking ability than GA. The flow diagram of the basic MA is shown in Figure 5. MA starts from initializing the population, operates on memes with evolutionary thought, generates new individuals using generating functions (e.g., crossover and variation operators, etc.), and finally forms new populations using updating functions (e.g., selection operators, etc.).

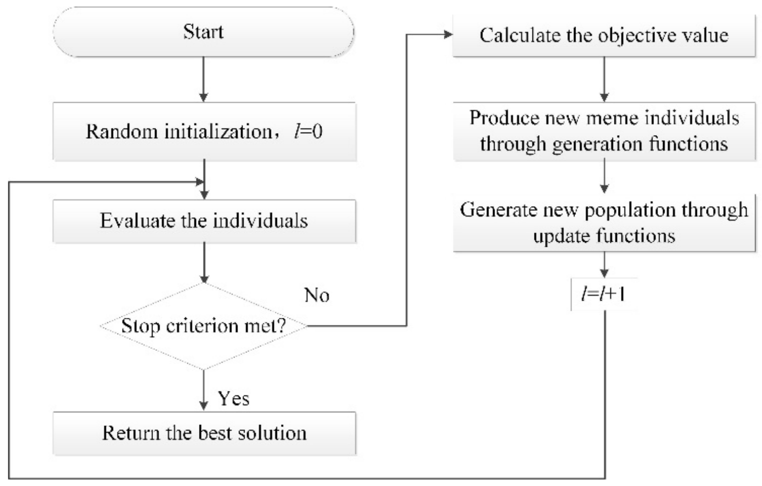
Start
Random initialization, l=0
Evaluate the individuals
Stop criterion met?
No
Yes
Return the best solution
Calculate the objective value
Produce new meme individuals through generation functions
Generate new population through update functions
l=l+1

Figure 5. Flow diagram of standard MA.

Although standard MA has a strong optimization-seeking capability, it also encounters problems such as insufficient global exploration capability [43]. On the other hand, DE has proven its powerful search capability since being proposed. Inspired by this, we embed the DE operators into MA and proposed a discrete MA (DMA). DMA mainly contains the following parts: population initialization, DE (mutation and crossover), and local search.

4.2. Population Initialization

In the initialization phase, we use the well-known job sequence-based method [9] to encode. The position of each job in the sequence represents its processing order on corresponding machines. An initialization method WPNEH (Weighted position NEH method, WPNEH) considering the weighted position of the job is proposed under the premise of determining the reordering jobs.

Step 1: Using the PFT_NEH(x) heuristic [9] and the initial schedule to generate two seeds π^P and π^B. The total weighted position of a single job is defined as follows:

$$\phi_j = \chi_1 \times \varphi_j(\pi^P) + \chi_2 \times \varphi_j(\pi^B) \tag{23}$$

In Equation (23), $\varphi_j(\pi^P)$ and $\varphi_j(\pi^B)$ represent the absolute positions of job j in two seeds. χ_1 and χ_2 represent the weight coefficients occupied by two seeds. The values of χ_1 and χ_2 are defined adaptively by the population size Ps and the order l ($l = 1, 2, \ldots, Ps$) of the newly generated individuals in the whole population:

$$\chi_1 = \frac{l}{Ps - 1} \tag{24}$$

$$\chi_2 = 1 - \frac{l}{Ps - 1} \tag{25}$$

Step 2: Arrange all jobs in decreasing order of the total weighted positions ϕ_j to obtain a sequence π^0.

Step 3: Create a new empty sequence π^{emp}. The jobs in π^0 are inserted into each position π^{emp} in turn. The solutions obtained from each insertion are evaluated based on Equations (11) and (12) to determine the optimal order. Continue the operations until all jobs finish insertion.

Step 4: Let $l = l + 1$, and continue steps 1–3 until all Ps individuals are generated.

The initialization procedure of WPNEH is shown as Algorithm 1.

Algorithm 1 WPNEH initialization

```
Input: job set $N_n$, population size $Ps$, initial schedule $B$ of a distributed factory
Output: initial population $\boldsymbol{POP}$
01:  Define the order of jobs in $N_n$, generate seed $\pi^B$
02:  Apply PFT_NEH(x) method to rearrange the jobs in $N_n$, generate seed $\pi^P$
03:  While $l \le Ps$ do
04:      Set the coefficient $\chi_1 = l/Ps - 1$ and $\chi_2 = 1 - l/(Ps-1)$
05:      Calculate $\phi_j$ for each job according to Equation (22)
06:      Generate a new sequence $\pi^0$ by arranging all jobs in descending order based on their $\phi_j$
07:      Execute the NEH insertion procedure, evaluate the solutions obtained, find the
best order
08:      Finish insertions of all jobs, obtain a new sequence $\pi^c$, count in $\boldsymbol{POP}$
09:      $l = l + 1$
10:  End While
```

4.3. DE Operation

DE [44] drives the search directions through the mutual synergistic and competitive behaviors among individuals of the population. Its overall structure is similar to GA, but the evolution principle is quite different. DE generates new individuals through perturbing the difference vectors of two or more individuals. It can obtain more operation space and enhance the global search capability when the individuals are significantly different from each other in the early stage of the search. In DMA, two key operators of DE (mutation and crossover) are embedded to perform the global search.

4.3.1. Mutation

The weighted difference vector of two individuals is first selected. Then, the weighted difference is summed with another individual vector. Hence, three individuals are randomly selected from the population ***POP***, the optimal one is defined as π_a, and the other two are defined as π_b and π_c. The mutation operator can be expressed as:

$$V_a = \pi_a \oplus \kappa \otimes (\pi_b - \pi_c) \tag{26}$$

where κ is the mutation scaling factor used to control the magnitude of the differences. "$\otimes$" represent the weighted differences between π_b and π_c:

$$\pi_a - \pi_b = \Delta \Leftrightarrow \delta(j) = \begin{cases} \pi_b(j), & \text{if } \pi_b(j) \neq \pi_c(j) \\ 0, & \text{otherwise} \end{cases} \quad j = 1, 2, \ldots, n \tag{27}$$

$$\kappa \otimes \Delta = \Phi \Leftrightarrow \varphi(j) = \begin{cases} \delta(j), & \text{if } rand() < \kappa \\ 0, & \text{otherwise} \end{cases} \tag{28}$$

In Equations (27) and (28), $\Delta = [\delta(1), \delta(2), \ldots, \delta(n)]$ and $\Phi = [\varphi(1), \varphi(2), \ldots, \varphi(n)]$ are the two temporary vectors used for the calculation. "$\oplus$" means the mutation individual π_V is obtained through adding Φ with the target individual π_{best}:

$$\pi_V = \pi_{best} \oplus \Phi \tag{29}$$

The generation process of π_V is described as follows.

Step 1: Select π_a, set $j = 1$.

Step 2: If $\varphi(j) = 0$, set $j = j + 1$, go to step 3; otherwise, generate a random number between (0, 1). If $rand() < \kappa$, update π_a by swapping the job $\pi_a(j)$ and $\varphi(j)$; else, insert the job $\pi_a(j)$ into all different positions of $\varphi(j)$, take the optimal solution and update π_a. Let $j = j + 1$.

Step 3: If $j \le n$, return to step 2; otherwise, return $\pi_V = \pi_a$.

The mutation procedure of DMA is sketched in Algorithm 2.

Algorithm 2 Mutation Operation

```
Input: population $\boldsymbol{POP}$, job number $n$, population size $Ps$, mutation factor $\kappa$,
temporary set $\Delta$ and $\Phi$
Output: mutation individual $\pi_V$
01:  Select 3 individuals ($\pi_a$, $\pi_b$ and $\pi_c$) from $\boldsymbol{POP}$ randomly
02:      For $j = 1$ to $n$ do
03:          Calculate the vector difference $\delta(j)$ between two individuals and save in $\Delta$
04:          Generate $rand()$ between (0,1), calculate the mutation difference $\varphi(j)$ and save in $\Phi$
05:    End For
06:    Output $\Phi$
07:    For $j = 1$ to $n$ do
08:        If $\varphi(j) = 0$
09:            $j = j + 1$
10:        Else
11:            generate $rand()$ between (0, 1)
12:                If $rand() < \kappa$
13:                    exchange job $\pi_a(j)$ and $\varphi(j)$ in $\pi_a$
14:                Else insert $\varphi(j)$ from $\pi_a$ into all positions of after $\pi_a(j)$, take the
optimal solution
15:                End If
16:            return the $\pi_a$
17:        End If
18:    End For
19:    Let $\pi_V = \pi_a$, output $\pi_V$
```

4.3.2. Crossover

When solving discrete scheduling problems based on job sequence, the probability factor is mainly used to determine the crossed jobs [38]. In this study, we improved the determination method of the crossed jobs by eliminating the crossover probability factor and proposed a random crossover operator: First, select two jobs randomly from the mutated individual π_V; second, put these two jobs and jobs between them in a temporary set N_{temp}; third, determine the positions in π_d of all jobs from N_{temp}, remove all jobs from N_{temp} and keep their positions. Finally, insert the jobs in π_d with original order to obtain a new sequence π'. The crossover process for π_d is the same. When the crossover operation of the two parents is completed, the new individual obtained is evaluated and the best one is retained. The crossover operation is sketched in Algorithm 3.

Algorithm 3 Crossover Operation

```
Input: mutation individual $\pi_V$, target individual $\pi_d$, temporary job set $N_{temp}$
Output: new individual $\pi_{new}$
01: # Crossover operation on $\pi_d$
02:  Select two jobs from $\pi_V$ randomly, put the jobs between them in $N_{temp}$ in turn
03:  Remove the jobs belonging to $N_{temp}$ from $\pi_d$, keep the vacant position unchanged
04:  Insert the jobs in $N_{temp}$ into the free positions of $\pi_d$ to obtain the new solution $\pi'$
05: # Crossover operation on $\pi_V$
06:  Ensure $\pi_d$ has the job $j \in N_{temp}$, clear $N_{temp}$, put $j$ in $N_{temp}$ orderly
07:  Remove jobs belonging to $N_{temp}$ from $\pi_V$, keep the vacant position unchanged
08:  Insert the jobs in $N_{temp}$ into the free positions of $\pi_d$ to obtain the new solution $\pi''$
09:  Evaluate new solutions:
10:      If $f(\pi'') < f(\pi')$
11:          Let $\pi_{new} = \pi''$, return $\pi_{new}$
12:      Else
13:          Let $\pi_{new} = \pi'$, return $\pi_{new}$
14:      End If
```

To facilitate understanding, **Example 4-1** presents the procedure of DE operation in detail.

Example 4-1

Mutation: Three individuals are randomly selected from the initial population: $\pi_a = [J_6, J_3, J_2, J_4, J_1, J_5]$, $\pi_b = [J_1, J_4, J_6, J_2, J_5, J_3]$ and $\pi_c = [J_3, J_4, J_2, J_1, J_5, J_6]$. With Equation (26) it yields $\Delta = \pi_b - \pi_c = [J_1, 0, J_6, J_2, 0, J_3]$. A set of random numbers $[0.7, 0.6, 0.9, 0.4, 0.1, 0.3]$ are generated according to Equation (27), so that the mutation scaling factor is $\kappa = 0.5$, then obtain $\Phi = [0, 0, 0, J_2, 0, J_3]$. It can be deduced that when $j = 4$ and $j = 6$, there are $\varphi(4) = J_2$ and $\varphi(6) = J_3$. For $j = 4$, a random number 0.2 (<0.5) is generated between the interval (0,1) satisfying the uniform distribution. Then, two jobs $\pi_a(4) = J_4$ and $\varphi(4) = J_2$ in π_a are swapped and a new solution $\pi_a = [J_6, J_3, J_4, J_2, J_1, J_5]$ is obtained; for $j = 6$, a random number 0.7 (>0.5) is also generated, the job $\varphi(6) = J_3$ in π_a is inserted into the latter position of $\pi_a(6) = J_5$ and a new solution $\pi_a = [J_6, J_4, J_2, J_1, J_5, J_3]$ is obtained.

Crossover: The mutation individual $\pi_V = [J_6, J_4, J_2, J_1, J_5, J_3]$ and the target individual $\pi_d = [J_5, J_3, J_2, J_4, J_6, J_1]$ are crossed as parents. First, two jobs J_2 and J_5 are randomly selected from π_V, and set $N_{temp} = [J_2, J_1, J_5]$. For π_d, remove the same jobs from N_{temp}, obtain $\pi_d = [X, J_3, X, J_4, J_6, X]$. The new solution is obtained by reinserting $N_{temp} = [J_2, J_1, J_5]$ into π_d. The derivation of the new solution for π_V is the same as π_d, and the result is $\pi_V = [J_6, J_4, J_5, J_2, J_1, J_3]$.

4.4. *Job Block-Based Random Reference Local Search*

As mentioned above, the two key operations of DE can improve the individuals concerning the vector difference of the population. Along with the iteration of an algorithm, the difference between individuals becomes smaller, which tends to lead the algorithm to local optimum easily. Therefore, DMA needs to equip with a local search framework to enhance its exploitation capability. For a long time, reference local search (RLS) has proved to be an effective local search algorithm and is often used to enhance the exploitation of metaheuristics [9]. RLS firstly generates a random reference sequence $\pi_r = [\pi_r(1), \pi_r(2), \ldots, \pi_r(z)]$, and uses it as a reference to guide the direction of the local search; subsequently, the jobs $\pi_r(j)$ are sequentially removed and inserted into all remaining positions of π_r to obtain new solutions. The optimal solution is compared with the incumbent solutions of the population, and if it is better, it is replaced with the population. The insertion process is repeated until all the jobs are traversed. Though RLS has a strong local exploitation capability, it still has some problems. On one hand, RLS uses a single job insertion operation, which may destroy the good information of incumbent solutions and cause the loss of other good solutions. On the other hand, the fixed order of the reference sequence and the fixed insertion process of the jobs will result in a fixed path of local search. If $\pi_r(j)$ is constant for a long time, it will cause a large number of repeated searches, which directly affects the search efficiency of the algorithm.

Boejko et al. [45] have pointed out that in job sorting scheduling problems, compound moves (insertion and swap) based on the job block can retain excellent sequence information during the evolution of an algorithm. It thus expands the neighborhood structure and search space, which is better than single job insertion and swap operations. Inspired by this idea, we hybridize RLS and compound moves of job block, and propose a random reference local search based on job block (BRRLS): firstly, generate a reference sequence $\pi_r = [\pi_r(1), \pi_r(2), \ldots, \pi_r(z)]$ randomly, where n_r represents the number of jobs to be rescheduled; secondly, select two jobs J_a and J_b randomly, construct the job block (including J_a and J_b) and take out all jobs in the individual π_{block} that needs local search; then, insert the job block into all possible positions of π, evaluate the generated solutions, and select the optimal one. Repeat the above procedure (each time select two unselected jobs) until all jobs are traversed. The BRRLS process is sketched in Algorithm 4.

Algorithm 4 BRRLS Procedure

```
Input: job set $N_n$, individual $\pi$, temporary set $N_{temp}$, temporary set $\Lambda$
Output: new individual $\pi_{new}$
01:   Randomly sort the jobs in $N_n$ to generate a reference sequence $\pi_r$
02:   Randomly select two jobs $J_a$ and $J_b$ from $N_n$
03:   Determine the block $\pi_{block}$ between $J_a$ and $J_b$ in $\pi_r$ (including $J_a$ and $J_b$), save $\pi_{block}$
in $N_{temp}$
04:   Remove all jobs belonging to $N_{temp}$ from $\pi$
05:   Insert $\pi_{block}$ in all positions of $\pi$, evaluate and select the optimal solution, save in $\Lambda$
06:   Clear $N_{temp}$, delete $J_a$ and $J_b$ from $N_n$
07:   Repeat the above operation (Line 03–07) until $len(N_n) \leq 1$
08:   Evaluate the individuals in $\Lambda$, return the optimal solution to $\pi_{new}$
```

To further improve the algorithmic performance, a simulated annealing-like (SA) mechanism [46] is introduced as an acceptance criterion for BRRLS, which guides DMA to receive a certain percentage of poor solutions during the search to avoid being trapped in local optimums. The idea of SA is to compare the neighborhood solution π' obtained by BRRLS with the incumbent solution π. If $f(\pi')$ is better than $f(\pi)$, SA replaces π with π', otherwise, the decision of whether to accept π' is based on a reception probability μ: A random number $rand()$ satisfying a uniform distribution is generated between $(0, 1)$; if $rand() < \mu$, π is replaced by the worse neighborhood solution obtained by the search. μ is expressed as follows:

$$\mu = e^{-(f(\pi')-f(\pi))/Temp} \tag{30}$$

where *Temp* represents the temperature constant:

$$Temp = T_0 \frac{\sum_{j=1}^{n_k} \sum_{i=1}^{m} P_{\pi_k(l),i}}{10 \times m \times n}, k \in \{1, 2, \ldots, f\} \tag{31}$$

In Equation (31), T_0 is the temperature adjustment parameter preset by SA. It can be seen from Equation (31) that the closer $f(\pi')$ is to $f(\pi)$, the closer the value μ is to 1 and π' will be accepted with a higher probability. Conversely, if $f(\pi')$ is much worse than $f(\pi)$, the value μ will be close to 0, and the solution π' will be dropped with a higher probability. Hence, the SA-based reception mechanism ensures that the population does not deviate from the current search position, but can additionally absorb a certain percentage of non-quality solutions to avoid the algorithm from falling into local optimums.

4.5. Update Strategy of the Population

To maintain the diversity of individuals, the following strategy is applied to update the population: first, a new individual is generated using the mutation and crossover operators; subsequently, a local search is performed on these individuals, and the individuals are updated. finally, the incumbent solutions are replaced by the newly generated solutions, and the uniqueness of these new solutions is ensured to complete a single iteration of the whole population.

4.6. Flowchart of DMA

According to previous descriptions, Algorithm 5 presents the flowchart of DMA.

The flow diagram of DMA is sketched in Figure 6. In general, DMA contains population initialization, DE operation, and local search. In the initialization phase, the population is generated using the WPNEH method; in the DE phase, the discrete differential mutation and crossover operators are executed to obtain the child individuals; in the local search phase, BRRLS is performed, and the simulated annealing mechanism is adopted as the reception criterion. Finally, the population is updated and the optimal solution is output.

Algorithm 5 Flowchart of DMA

```
Input: population size $Ps$, mutation scaling factor $\kappa$, reordered job number $n$
Output: best solution $\pi$
01: While termination condition not met do
02:   # Initialization (Section 4.2)
03:       Use WPNEH method to create new individuals
04:       Construct POP, evaluate the individuals
05:   # DE (Section 4.3)
06:       Perform DE according to $\kappa$, generate $Ps/2$ individuals
07:       Perform random crossover operation on mutated individuals, and evaluate
08:   # BRRLS (Section 4.4)
09:       Implement the SA-based BRRLS procedure on the $Ps/2$ individuals
10:       Replace the incumbent solutions using solutions obtained by local search
11:       Update the population (Section 4.5)
12: End While
```

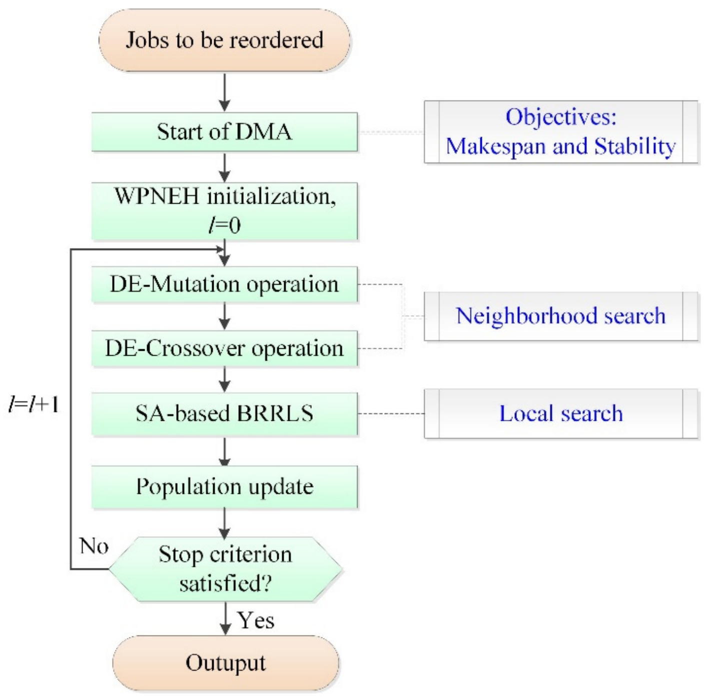

Figure 6. Flow diagram of DMA.

5. Experimental Comparison and Analysis

5.1. Experimental Settings

Since few pieces of literature and public benchmarks have been developed for DDBFSP, we apply DFOA on its benchmark [8] to obtain test instances. These test instances are used as the initial schedules for each distributed factory. To fulfill different experimental requirements, the range of variable intervals of the DPFSP benchmark is set as $n \in \{50, 100, 200\}$, $m \in \{5, 10, 20\}$ and $f \in \{2, 3, 4\}$. There are 27 combinations of different parameters, each containing 10 instances. The termination criterion of the algorithm is set to $T_{\max} = 90 \times n \times m$ milliseconds.

The breakdown events are simulated according to the mechanism introduced in Section 2.2. When an event occurs on machine i, the trigger node is firstly limited to the interval $[P_{\pi_k(l),i}, C_{\pi_k(l),i}]$ to ensure the timeliness of the breakdown. Since DMA performs only on partial jobs which have not started processing, we compress the breakdown interval to $[P_{\pi_k(l),i}, C_{\pi_k(n_k-2),1}]$, i.e., the start time of processing to the completion time of the penultimate job at the first machine, to ensure a feasible execution space for DMA.

The experiments are conducted on a PC with Intel(R) Core(TM) i7-8700 CPU and 16G RAM configuration, and the involved programs are compiled by Python. To balance the objective functions (makespan and stability), both weight coefficients w_1 and w_2 are set to 0.5. The algorithm is repeated 10 times for each breakdown in each factory, and the experiments use the average relative percent deviation (ARPD) as the metric to evaluate the mean quality of the obtained solutions. Since DMA is implemented for each distributed factory, the sum of ARPDs of all factories is firstly calculated, and the mean value is defined as the ARPD value for a single case of DDBFSP. The experiments will be conducted in the following three perspectives:

(1) Key parameter calibration;
(2) Effectiveness of the proposed optimization strategy;
(3) Comparison with other intelligent algorithms.

5.2. Parameter Calibration

DMA contains three key parameters: population size Ps, mutation scaling factor κ, and temperature adjustment parameter T_0. The design of the experiment (DOE) [9] method is used for parameter calibrations, and in total 36 sets of initial schedules with factory number $f = 3$ were generated. The number of random breakdowns for each distributed factory is set to 2. The key parameters and ARPD values are defined as the control factors and response variables, respectively. The candidate values of the above parameters are set as: $Ps = \{30, 50, 80, 100\}$, $\kappa = \{0.4, 0.7, 0.9\}$ and $T_0 = \{0.1, 0.2, 0.3, 0.4\}$, which generate 48 configuration combinations. We use the "Analysis of Variance" (ANOVA) method to analyze the statistical results, as shown in Table 1.

Table 1. ANOVA results of DMA parameter combinations.

Source	Sum of Squares	Degree of Freedom	Mean Square	*F*-Ratio	*p*-Value
Ps	38.4	3	12.8	134.4	**0.007**
κ	144.3	2	72.2	382.4	**0.000**
T_0	6.8	3	2.3	17.6	**0.012**
$Ps \times \kappa$	17.29	6	2.9	8.8	0.354
$Ps \times T_0$	4.8	9	0.5	0.55	0.492
$\kappa \times T_0$	0.5	6	0.1	2.46	0.087

As can be seen in Table 1, the p-values of Ps, κ and T_0 are all less than 0.05 confidence level, which means all the parameters have important impacts on the performance of DMA. Among them, the corresponding F-ratio value of κ is the largest, which indicates that κ has the greatest impact on DMA. Moreover, Table 1 shows that the p-values of the interactions between every two parameters are greater than 0.05, which means the parameter interactions do not have a significant effect on DMA, and the parameter can be selected directly from the main effects plot in Figure 7.

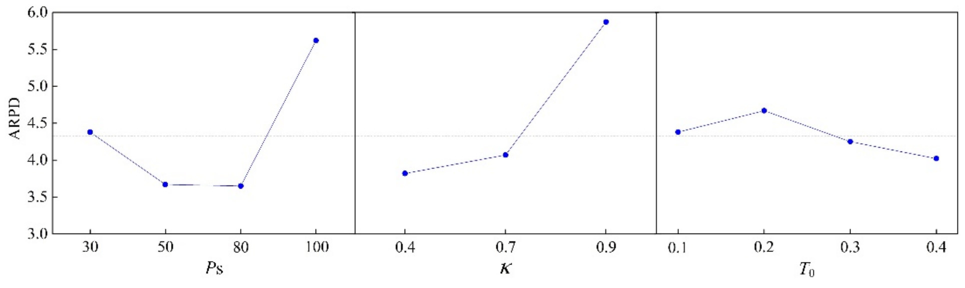

Figure 7. Main effects plot of the parameters for DMA.

From Figure 7, it can be observed that the performance of DMA decreases with the increment of κ, and DMA obtains the best results when $\kappa = 0.4$. A relatively overlarge mutation scaling factor increases the randomness of search and leads to degradation of the mutation. The effect of Ps ranks second, the performance of DMA first increases as the number of Ps increases, it starts to decrease after reaching the optimum. This indicates that increasing Ps appropriately will enhance the diversity of the population. However, an overlarge Ps consumes too much running time of a single iteration, which in turn compresses the number of iterations and leads to a decrease in the probability of obtaining the optimal solution. The effect of T_0 on the algorithmic performance ranked the 3rd, and the main effect plot shows that the performance fluctuates with the growth of T_0, DMA obtained the best results when $T_0 = 0.4$. Based on the above analysis, the parameter combinations of DMA are set as: $Ps = 80$, $\kappa = 0.4$, $T_0 = 0.4$.

5.3. Effectiveness of the Proposed Algorithmic Component

DMA contains three important components: WPNEH initialization, DE operators, and BRRLS. To verify their effectiveness, we mask one corresponding part of DMA each time, and generate three types of variant algorithms from DMA: (1) DMA_RI: random initialization, which is used to verify the effectiveness of WPNEH initialization; (2) DMA_ND: without neighborhood search, which is used to verify the effectiveness of DE operators; (3) DMA_NL: without BRRLS, which is used to verify the effectiveness of BRRLS. The parameter settings and instances used in Section 5.2 were adopted. As the randomness of breakdown has a large impact on the results, to ensure the fairness of the comparison, only one complete set of breakdowns is simulated, and all variant algorithms are tested under this scenario. Table 2 shows the comparison results.

Table 2. Comparison results between DMA and its variant algorithms.

Instance	**ARPD**			
$n \times m \times f$	**DMA_RI**	**DMA_ND**	**DMA-NL**	**DMA**
50 × 5 × 3	0.56	1.24	0.92	**0.54**
50 × 10 × 3	0.61	1.55	0.98	**0.36**
50 × 20 × 3	1.17	1.73	1.12	**0.48**
100 × 5 × 3	1.33	2.26	2.60	**0.18**
100 × 10 × 3	2.05	2.44	2.38	**0.11**
100 × 20 × 3	2.84	3.23	3.15	**0.05**
200 × 5 × 3	3.97	5.58	4.19	**0.00**
200 × 10 × 3	4.63	6.67	5.49	**0.00**
200 × 20 × 3	4.17	6.04	5.32	**0.00**

From Table 2, we can observe that the performance of DMA outperforms the other variants in all scenarios. In specific analysis, DMA outperforms DMA_RI, representing that the WPNEH initialization strategy can provide a better search starting point for DMA; DMA outperforms DMA_ND, indicating that the DE operators can improve the performance of DMA effectively. The results of DMA_NL are inferior to those of DMA, which means that the proposed BRRLS and SA-based reception criterion have an important influence on the optimization. BRRLS has retained the "greedy search" idea from RLS but improved the selection of jobs in the reference sequence. It ensures the inconsistency of local search step length through random selection of job blocks strategy, which makes the solutions obtained by local search more diverse and helps DMA to jump out of the local optimum. On the other hand, it can be observed from Table 2 that the differences between the compared algorithms are not significant when the scale of the instance is relatively small (e.g., $n = 50$). If the processing tasks (number of jobs) assigned to a distributed factory are small, the corresponding reorder execution space is also smaller, and the comparison algorithms are more likely to obtain optimal or suboptimal solutions in a given time. With the gradual increase in the problem size, the differences between algorithms start to manifest. The performance of each variant algorithm decreases more on the large-scale instances, while DMA remains relatively more stable.

To verify whether the differences were significant, we conducted a significance test from a statistical point of view. Each algorithm and ARPD value were chosen as the control variable and the response variable. Figure 8 demonstrates the mean plot of the 95% confidence interval. As can be seen, the ARPD values of each algorithm are ranked from top to bottom as DMA_ND, DMA_RI, DMA_NL, and DMA. The confidence intervals of each two algorithms do not overlap, which indicates that DMA is significantly better than its variants. The experimental results reveal that the proposed optimization strategies for each search phase jointly ensure the performance of DMA.

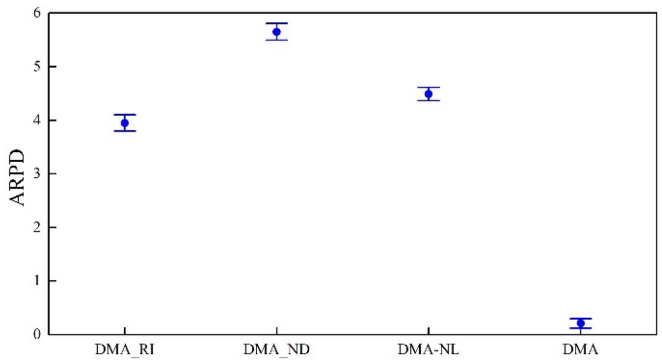

Figure 8. Mean plots with 95% confidence intervals for DMA and its variant algorithms.

5.4. Comparison with Other Intelligent Algorithms

In this section, DMA is compared with the algorithms for solving traditional flow shop rescheduling problems. The compared algorithms are (1) Iterative Local Insertion Search (ILS) [25]; (2) Iterative Greedy Algorithm (IG) [25]; (3) Improved Migratory Bird Algorithm (IMBO) [47]; (4) Discrete Teaching and Learning Optimization (DTLO) Algorithm [29]. We follow strictly the literature to compile all the comparison algorithms, which share the same data structure, objective function, and termination criterion. To ensure comparison fairness, we apply the same rescheduling process, i.e., the comparison algorithms share the same initial schedules, the same time node distribution of breakdown events, and the same rescheduling strategy. The algorithms are only used to reorder unprocessed jobs as a way to test their optimization efficiencies. The calculation of ARPD is consistent with previous sections. The parameter combinations of the comparison algorithm were pre-adjusted and the specific settings are shown in Table 3. According to [25], as a single local search algorithm, ILS stops immediately after obtaining a local optimum, so no special parameters or termination conditions need to be set for ILS.

Table 3. Parameter determination of the compared algorithms.

Algorithm	**Parameters**
ILS	--
IG	Job numbers of destruction 4
IMBO	Population size 50; Neighborhood set size 7; Migratory birds pass on the number of solutions 3; Number of lead bird iterations 20; Number of population iteration 100; weighted factor 0.8
DTLO	Population size 50; learning factor of teacher 1

Table 4 shows the statistical results of different instances under single breakdown ($\beta = 1$), the optimal values were bolded. It can be seen that DMA outperforms other algorithms in all instances of distributed scenarios. DTLO, IG, IMBO, and ILS obtain results in 2nd, 3rd, 4th and 5th place. It indicates that DMA is more suitable as a reordering algorithm. Specifically, ILS is an algorithm containing only a local insertion search framework, which lacks key structures such as population initialization and neighborhood search. It is difficult for ILS to balance exploration and exploitation, and therefore has the worst performance. As metaheuristics, DTLO, IG, and IMBO are more competitive in terms of performance on small-scale instances. For example, when the number of the distributed factory is set as $f = 4$, and the total number of jobs is $n = 50$, the results of other algorithms do not differ from DMA. The main reason is that when the size of the initial schedule is small, fewer processing tasks are assigned to each distributed factory. The search space for the solution becomes smaller, so the efficiency of each algorithm in finding the best solutions in a given time becomes high, thus reducing the variability of the comparison results. The same phenomenon is reflected in the scenarios of $\beta = 2$ and $\beta = 3$, as shown in Tables 5 and 6. DMA also performs the best among all comparison algorithms. As the number of breakdowns increases, the performances of the compared algorithms are not affected too much by small-scale instances. The performances on large-scale instances ($n = 200$) showed different degrees of degradation. The errors of single rescheduling accumulate with the increment of breakdowns, which causes a deterioration effect, and thus decreases the algorithmic performances relatively. In comparison, the statistical results of DMA under different breakdown scenarios are less different, which also reveals that DMA is more stable and robust on both small- and large-scale instances.

Table 4. Comparison results of different algorithms by $\beta = 1$.

Initial Instance	**ARPD**				
$n \times m \times f$	**ILS**	**IG**	**IMBO**	**DTLO**	**DMA**
50 × 5 × 2	2.40	1.46	1.43	1.06	**1.02**
50 × 5 × 3	2.18	0.55	0.85	0.51	**0.39**
50 × 5 × 4	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**
50 × 10 × 2	2.93	1.77	1.48	1.13	**1.08**
50 × 10 × 3	2.46	0.72	0.83	0.66	**0.28**
50 × 10 × 4	0.14	**0.00**	**0.00**	**0.00**	**0.00**
50 × 20 × 2	3.17	1.04	1.47	1.19	**1.05**
50 × 20 × 3	2.89	0.85	1.04	0.69	**0.34**
50 × 20 × 4	1.74	0.31	0.49	**0.00**	**0.00**
100 × 5 × 2	4.23	2.53	2.28	2.24	**0.53**
100 × 5 × 3	3.44	1.56	1.71	1.47	**0.17**
100 × 5 × 4	2.56	1.42	1.38	1.25	**0.08**
100 × 10 × 2	4.49	2.06	2.33	2.09	**0.58**
100 × 10 × 3	3.23	1.60	1.63	1.71	**0.22**
100 × 10 × 4	2.91	1.53	1.24	1.39	**0.13**
100 × 20 × 2	5.44	2.55	2.70	2.31	**0.32**
100 × 20 × 3	3.69	1.97	1.80	1.88	**0.04**
100 × 20 × 4	3.28	1.11	1.31	1.78	**0.37**
200 × 5 × 2	5.78	2.88	3.53	2.90	**0.00**
200 × 5 × 3	4.95	2.19	2.61	2.13	**0.12**
200 × 5 × 4	4.16	2.28	2.39	2.11	**0.08**
200 × 10 × 2	5.57	3.54	3.72	3.02	**0.06**
200 × 10 × 3	5.11	2.05	2.47	2.00	**0.00**
200 × 10 × 4	4.39	2.29	2.45	2.03	**0.00**
200 × 20 × 2	6.06	2.98	3.84	3.23	**0.00**
200 × 20 × 3	6.08	2.57	2.79	1.97	**0.00**
200 × 20 × 4	4.47	2.56	2.94	2.19	**0.00**
Ave.	3.62	1.71	1.87	1.59	**0.25**

To further verify the superiority of DMA, the differences between the compared algorithms are observed through statistical tests. ANOVA is used to describe the mean plot with a 95% confidence interval of the results obtained by the algorithms for different f, as shown in Figure 9.

It can be observed that the ARPD of DMA falls below that of other algorithms, and none of the confidence intervals overlap. It indicates again that the optimization performance of DMA is significantly better than its comparators. Moreover, it can be seen from Figure 9 that the performance of each algorithm gradually becomes better as f increases.

This is mainly because the reordering algorithm does not focus on the assignment of jobs to plants in the initial schedule, rather on the reordering within the distributed factories. An increase in f leads to a decrease in assigned jobs and a corresponding decrease in their computational complexity and, therefore, an increase in the optimization-seeking efficiency of an algorithm.

Moreover, we compared and analyzed the performances under different numbers of breakdowns (β) based on the statistical results. Figure 10 shows the performance curves of the comparative algorithms. It can be seen that ARPD values increase as β gradually increases, representing that all the algorithms are affected by the frequency of breakdowns. Compared with other algorithms, ARPDs of DMA are improved by at least 88%. Moreover, ARPD values of ILS, IG, IMBO, and DTLO algorithms fluctuate more and show an increasing trend with the increase in β. Comparatively, ARPD values of DMA fluctuate the least, which indicates that the robustness of DMA under different scenarios is better than the compared algorithms.

In summary, DMA has a good performance for local reordering. Its innovation and advantages can be summarized as follows: (1) WPNEH initialization method provides a better initial population and high-quality search starting point for DMA; (2) the mutation and crossover operators based on DE provides an excellent neighborhood search capability; (3) BRRLS provides stronger local exploitation; (4) the SA-based reception criterion can help DMA jump out of the local optimum effectively. Among them, (2), (3) and (4) balance the exploration and exploitation of DMA.

Table 5. Comparison results of different algorithms by $\beta = 2$.

Initial Instance	**ARPD**				
$n \times m \times f$	**LS**	**IG**	**IMBO**	**DTLO**	**DMA**
50 × 5 × 2	2.89	1.66	1.82	1.45	**0.83**
50 × 5 × 3	2.06	0.41	0.52	0.28	**0.25**
50 × 5 × 4	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**
50 × 10 × 2	3.04	1.77	1.84	1.50	**0.98**
50 × 10 × 3	2.11	0.32	0.63	0.26	**0.13**
50 × 10 × 4	0.22	**0.00**	**0.00**	**0.00**	**0.00**
50 × 20 × 2	3.28	1.74	1.90	1.56	**0.95**
50 × 20 × 3	2.04	0.36	0.61	0.33	**0.21**
50 × 20 × 4	1.65	0.55	0.53	**0.00**	**0.00**
100 × 5 × 2	3.55	2.76	2.87	2.51	**0.68**
100 × 5 × 3	3.79	1.84	1.97	1.59	**0.19**
100 × 5 × 4	2.74	1.71	1.69	1.28	**0.84**
100 × 10 × 2	3.92	2.66	2.75	2.44	**0.55**
100 × 10 × 3	3.72	1.93	1.93	1.64	**0.31**
100 × 10 × 4	2.63	1.58	1.82	1.35	**0.67**
100 × 20 × 2	4.73	2.85	2.92	2.67	**0.41**
100 × 20 × 3	3.96	1.85	2.05	1.78	**0.18**
100 × 20 × 4	3.04	1.66	1.92	1.20	**0.23**
200 × 5 × 2	6.12	4.03	4.73	4.09	**0.12**
200 × 5 × 3	6.58	3.04	3.67	2.82	**0.05**
200 × 5 × 4	4.16	2.84	3.04	2.55	**0.08**
200 × 10 × 2	6.49	4.39	4.54	4.15	**0.00**
200 × 10 × 3	6.71	3.05	3.44	2.96	**0.00**
200 × 10 × 4	4.39	2.65	3.45	2.91	**0.13**
200 × 20 × 2	6.88	4.45	4.96	4.32	**0.00**
200 × 20 × 3	6.33	3.58	3.70	2.71	**0.00**
200 × 20 × 4	4.47	2.43	3.49	2.63	**0.00**
Ave.	3.75	2.07	2.33	1.88	**0.29**

Table 6. Comparison results of different algorithms by $\beta = 3$.

Initial Instance	ARPD				
$n \times m \times f$	LS	IG	IMBO	DTLO	DMA
50 × 5 × 2	1.80	0.58	0.73	0.39	**0.20**
50 × 5 × 3	2.01	0.34	0.64	0.28	**0.15**
50 × 5 × 4	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**
50 × 10 × 2	2.93	1.70	1.88	1.47	**0.11**
50 × 10 × 3	2.17	0.45	0.70	0.30	**0.24**
50 × 10 × 4	1.31	0.00	0.00	0.00	**0.00**
50 × 20 × 2	3.28	2.01	2.03	1.92	**0.06**
50 × 20 × 3	1.94	0.63	0.78	0.44	**0.17**
50 × 20 × 4	1.59	0.24	0.43	**0.00**	**0.00**
100 × 5 × 2	4.37	3.04	3.15	2.83	**0.05**
100 × 5 × 3	3.65	2.19	2.24	2.02	**0.19**
100 × 5 × 4	2.93	2.11	2.18	1.85	**0.00**
100 × 10 × 2	4.62	2.96	3.29	2.89	**0.52**
100 × 10 × 3	3.98	2.73	2.81	2.50	**0.28**
100 × 10 × 4	3.15	2.06	2.86	1.72	**0.00**
100 × 20 × 2	5.89	3.12	3.56	2.84	**0.13**
100 × 20 × 3	4.71	3.17	3.37	3.01	**0.06**
100 × 20 × 4	3.44	2.88	2.94	1.56	**0.00**
200 × 5 × 2	6.61	4.09	4.91	3.47	**0.00**
200 × 5 × 3	6.23	4.61	5.23	2.98	**0.00**
200 × 5 × 4	6.16	3.77	4.53	1.93	**0.00**
200 × 10 × 2	6.79	4.61	4.86	3.54	**0.00**
200 × 10 × 3	6.47	4.87	5.65	3.94	**0.00**
200 × 10 × 4	6.57	4.19	4.29	2.20	**0.00**
200 × 20 × 2	6.84	5.12	5.27	4.07	**0.00**
200 × 20 × 3	6.97	5.43	5.19	4.83	**0.00**
200 × 20 × 4	6.21	3.98	4.84	2.58	**0.00**
Ave.	4.16	2.63	2.94	2.04	**0.08**

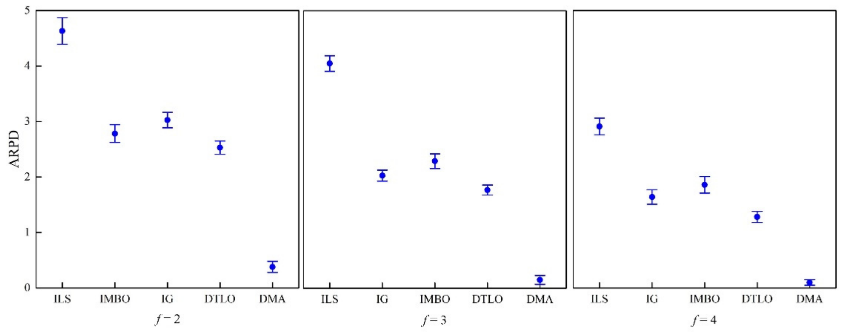

Figure 9. Mean plot with 95% confidence interval of the compared algorithms on different scenarios.

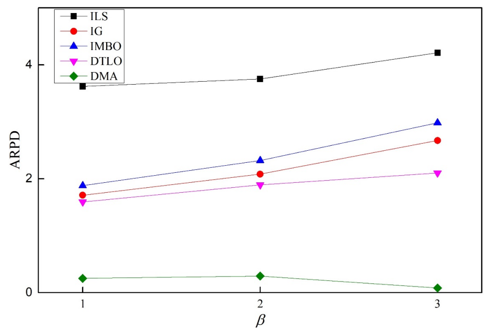

Figure 10. Performance comparison with different breakdown numbers.

6. Conclusions

Building rescheduling optimization models and designing effective optimization methods according to the characteristics of distributed manufacturing are of significance to promote the research of the dynamic scheduling theory of distributed manufacturing. This study investigated the rescheduling strategy and algorithm for DDBFSP, in which machine breakdown events are considered as the disruption in the manufacturing site. Firstly, the mathematical model of DDBFSP including the event simulation mechanism is constructed. We consider makespan and stability as the objectives. The goal of this study is to optimize the bi-objective when the stochastic breakdown occurs in any distributed factories. We apply the "event-driven" policy in response to the disruption. A two-stage "predictive-reactive" rescheduling strategy is proposed. In the first stage, a static environment (DBFSP) without machine breakdown is considered, and the global initial schedules are generated; in the second stage, after machine breakdown occurs, the initial schedule is locally optimized by a hybrid repair policy based on "right-shift repair + local reorder", and the DMA reordering algorithm based on DE is proposed for local reorder operation. For DMA, a WPNEH initialization method is designed to generate a high-quality population. In the neighborhood search phase, DE is embedded to improve the neighborhood structure and expand the target space by using mutation and crossover operators; in the local search phase, the BRRLS framework is proposed to perturb the high-quality solutions. To maintain the diversity, BRRLS has combined with the SA mechanism to receive the worse solutions with a certain probability. To obtain the best performance of DMA, the DOE method is used to calibrate three key parameters. The effectiveness of the proposed optimization strategy for DMA is verified through comparative experiments. Finally, DMA is compared with other algorithms on different test instances. The statistical analysis using ANOVA has verified the superiority of DMA.

Although the proposed rescheduling strategy has shown effectiveness, there still exist shortcomings. In this study, we only considered the breakdown event as the disruption. Real-life manufacturing suffers from far more than one disruption. The other common disruptions such as job cancellations and their interaction mechanism should be deeply investigated. Therefore, future works will concentrate on the construction of a more refined model that can manage more disruptions simultaneously.

This study attempts to explore the dynamic scheduling problem from the perspective of operational research optimization. With the development of the Industrial 4.0 network and big data, other artificially intelligent technologies play increasingly important roles in smart manufacturing. Combining data-driven technology with intelligent algorithms could adopt their respective advantages, and create more advanced optimization frameworks. For example, intelligent optimization can provide a large amount of historical scheduling data, which can be aggregated with other industrial information as a sample source for data-driven and machine learning. Therefore, the scheduling decision-making function can be deployed hierarchically and decoupled according to different scenarios and environments, thus making rational use of computing resources and improving the flexibility and stability of the system.

Author Contributions: Conceptualization, Z.L.; methodology, X.Z.; software, Y.H.; validation, Z.L., X.Z. and M.R.; formal analysis, G.K.; investigation, R.S.; resources, G.K.; data curation, X.Z.; writing—original draft preparation, X.Z. and Y.H.; writing—review and editing, Z.L.; visualization, M.R.; supervision, G.K.; project administration, R.S.; funding acquisition, Z.L. All authors have read and agreed to the published version of the manuscript.

Funding: This research is funded by the Natural Science Foundation of Xuzhou, China (KC21070), National Natural Science Foundation of China (61803192), and the Narodowego Centrum Nauki, Poland (No. 2020/37/K/ST8/02748 & No. 2017/25/B/ST8/00962).

Data Availability Statement: All data can be requested from the corresponding author.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Helu, M.; Sobel, W.; Nelaturi, S.; Waddell, R.; Hibbard, S. Industry Review of Distributed Production in Discrete Manufacturing. *J. Manuf. Sci. Eng.* **2020**, *142*, 110802. [CrossRef]
2. Cheng, Y.; Bi, L.; Tao, F.; Ji, P. Hypernetwork-based manufacturing service scheduling for distributed and collaborative manufacturing operations towards smart manufacturing. *J. Intell. Manuf.* **2020**, *31*, 1707–1720. [CrossRef]
3. Valilai, O.F.; Houshmand, M. A collaborative and integrated platform to support distributed manufacturing system using a service-oriented approach based on cloud computing paradigm. *Robot. Comput.-Integr. Manuf.* **2013**, *29*, 110–127. [CrossRef]
4. Chen, S.; Pan, Q.K.; Gao, L. Production scheduling for blocking flowshop in distributed environment using effective heuristics and iterated greedy algorithm. *Robot. Comput.-Integr. Manuf.* **2021**, *71*, 102155. [CrossRef]
5. Ribas, I.; Companys, R.; Tort-Martorell, X. Efficient heuristics for the parallel blocking flow shop scheduling problem. *Expert Syst. Appl.* **2017**, *74*, 41–54. [CrossRef]
6. Ribas, I.; Companys, R.; Tort-Martorell, X. An iterated greedy algorithm for solving the total tardiness parallel blocking flow shop scheduling problem. *Expert Syst. Appl.* **2019**, *121*, 347–361. [CrossRef]
7. Zhang, G.; Xing, K.; Cao, F. Discrete differential evolution algorithm for distributed blocking flowshop scheduling with makespan criterion. *Eng. Appl. Artif. Intell.* **2018**, *76*, 96–107. [CrossRef]
8. Zhang, X.; Liu, X.; Tang, S.; Królczyk, G.; Li, Z. Solving Scheduling Problem in a Distributed Manufacturing System Using a Discrete Fruit Fly Optimization Algorithm. *Energies* **2019**, *12*, 3260. [CrossRef]
9. Shao, Z.; Pi, D.; Shao, W. Hybrid enhanced discrete fruit fly optimization algorithm for scheduling blocking flow-shop in distributed environment. *Expert Syst. Appl.* **2020**, *145*, 113147. [CrossRef]
10. Li, W.; Li, J.; Gao, K.; Han, Y.; Niu, B.; Liu, Z.; Sun, Q. Solving robotic distributed flowshop problem using an improved iterated greedy algorithm. *Int. J. Adv. Robot. Syst.* **2019**, *16*, 1729881419879819. [CrossRef]
11. Zhao, F.; Zhao, L.; Wang, L.; Song, H. An ensemble discrete differential evolution for the distributed blocking flowshop scheduling with minimizing makespan criterion. *Expert Syst. Appl.* **2020**, *160*, 113678. [CrossRef]
12. Miyata, H.H.; Nagano, M.S. The blocking flow shop scheduling problem: A comprehensive and conceptual review. *Expert Syst. Appl.* **2019**, *137*, 130–156. [CrossRef]
13. Chen, Q.; Deng, L.-F.; Wang, H.-M. Optimization of multi-task job-shop scheduling based on uncertainty theory algorithm. *Int. J. Simul. Model.* **2018**, *17*, 543–552. [CrossRef]
14. Liu, W.; Jin, Y.; Price, M. New meta-heuristic for dynamic scheduling in permutation flowshop with new order arrival. *Int. J. Adv. Manuf. Technol.* **2018**, *98*, 1817–1830. [CrossRef]
15. Chen, J.; Wang, M.; Kong, X.-T.; Huang, G.-Q.; Dai, Q.; Shi, G. Manufacturing synchronization in a hybrid flowshop with dynamic order arrivals. *J. Intell. Manuf.* **2019**, *30*, 2659–2668. [CrossRef]
16. Zhang, B.; Pan, Q.-K.; Gao, L.; Zhang, X.-L.; Peng, K.K. A multi-objective migrating birds optimization algorithm for the hybrid flowshop rescheduling problem. *Soft Comput.* **2019**, *23*, 8101–8129. [CrossRef]
17. Moghaddam, S.-K.; Saitou, K. Predictive-Reactive Rescheduling for New Order Arrivals with Optimal Dynamic Pegging. In Proceedings of the 2020 IEEE 16th International Conference on Automation Science and Engineering, Hong Kong, China, 20–21 August 2020; pp. 710–715.
18. Han, Y.; Gong, D.; Jin, Y.; Pan, Q. Evolutionary Multiobjective Blocking Lot-Streaming Flow Shop Scheduling With Machine Breakdowns. *IEEE Trans. Syst. Man Cybern.* **2019**, *49*, 184–197. [CrossRef] [PubMed]
19. Nica, E.; Stan, C.-I.; Luțan (Petre), A.-G.; Oașa (Geambazi), R.-Ș. Internet of Things-based Real-Time Production Logistics, Sustainable Industrial Value Creation, and Artificial Intelligence-driven Big Data Analytics in Cyber-Physical Smart Manufacturing Systems. *Econ. Manag. Financ. Mark.* **2021**, *16*, 52–62.
20. Popescu, G.-H.; Petreanu, S.; Alexandru, B.; Corpodean, H. Internet of Things-based Real-Time Production Logistics, Cyber-Physical Process Monitoring Systems, and Industrial Artificial Intelligence in Sustainable Smart Manufacturing. *J. Self-Gov. Manag. Econ.* **2021**, *9*, 52–62.
21. Cohen, S.; Macek, J. Cyber-Physical Process Monitoring Systems, Real-Time Big Data Analytics, and Industrial Artificial Intelligence in Sustainable Smart Manufacturing. *Econ. Manag. Financ. Mark.* **2021**, *16*, 55–67.
22. Vieira, G.-E.; Herrmann, J.-W.; Lin, E. Rescheduling manufacturing systems: A framework of strategies, policies, and methods. *J. Sched.* **2003**, *6*, 39–62. [CrossRef]
23. Framinan, J.-M.; Fernandez-Viagas, V.; Perez-Gonzalez, P. Using real-time information to reschedule jobs in a flowshop with variable processing times. *Comput. Ind. Eng.* **2019**, *129*, 113–125. [CrossRef]
24. Katragjini, K.; Vallada, E.; Ruiz, R. Flow shop rescheduling under different types of disruption. *Int. J. Prod. Res.* **2013**, *51*, 780–797. [CrossRef]
25. Iris, Ç.; Lam, J.-S.-L. Recoverable robustness in weekly berth and quay crane planning. *Transp. Res. Part B Methodol.* **2019**, *122*, 365–389. [CrossRef]
26. Ma, Z.; Yang, Z.; Liu, S.; Song, W. Optimized rescheduling of multiple production lines for flowshop production of reinforced precast concrete components. *Autom. Constr.* **2018**, *95*, 86–97. [CrossRef]
27. Li, J.; Pan, Q.; Mao, K. A hybrid fruit fly optimization algorithm for the realistic hybrid flowshop rescheduling problem in steelmaking systems. *IEEE Trans. Autom. Sci. Eng.* **2016**, *13*, 932–949. [CrossRef]

28. Li, J.; Pan, Q.; Mao, K. A discrete teaching-learning-based optimization algorithm for realistic flowshop rescheduling problems. *Eng. Appl. Artif. Intell.* **2015**, *37*, 279–292. [CrossRef]
29. Valledor, P.; Gomez, A.; Priore, P.; Puente, J. Solving multi-objective rescheduling problems in dynamic permutation flow shop environments with disruptions. *Int. J. Prod. Res.* **2018**, *56*, 6363–6377. [CrossRef]
30. Wade, K.; Vochozka, M. Artificial Intelligence Data-driven Internet of Things Systems, Sustainable Industry 4.0 Wireless Networks, and Digitized Mass Production in Cyber-Physical Smart Manufacturing. *J. Self-Gov. Manag. Econ.* **2021**, *9*, 48–60.
31. Hamilton, S. Real-Time Big Data Analytics, Sustainable Industry 4.0 Wireless Networks, and Internet of Things-based Decision Support Systems in Cyber-Physical Smart Manufacturing. *Econ. Manag. Financ. Mark.* **2021**, *16*, 84–94.
32. Riley, C.; Vrbka, J.; Rowland, Z. Internet of Things-enabled Sustainability, Big Data-driven Decision-Making Processes, and Digitized Mass Production in Industry 4.0-based Manufacturing Systems. *J. Self-Gov. Manag. Econ.* **2021**, *9*, 42–52.
33. Pelau, C.; Dabija, D.-C.; Ene, I. What Makes an AI Device Human-Like? The Role of Interaction Quality, Empathy and Perceived Psychological Anthropomorphic Characteristics in the Acceptance of Artificial Intelligence in the Service Industry. *Comput. Hum. Behav.* **2021**, *122*, 106855. [CrossRef]
34. Richard, D. *The Selfish Gene*; Oxford University Press: New York, NY, USA, 1976.
35. Wang, J.; Zhou, Y.; Wang, Y.; Zhang, J.; Chen, C.; Zheng, Z. Multiobjective vehicle routing problems with simultaneous delivery and pickup and time windows: Formulation, instances, and algorithms. *IEEE Trans. Cybern.* **2015**, *46*, 582–594. [CrossRef] [PubMed]
36. Decerle, J.; Grunder, O.; El Hassani, A.-H.; Barakat, O. A memetic algorithm for multi-objective optimization of the home health care problem. *Swarm Evol. Comput.* **2019**, *44*, 712–727. [CrossRef]
37. Yancey, S.-K.; Tsvetkov, P.-V.; Jarrell, J.-J. A greedy memetic algorithm for a multiobjective dynamic bin packing problem for storing cooling objects. *J. Heuristics* **2019**, *25*, 1–45.
38. Zhou, Y.; Fan, M.; Ma, F.; Xu, X.; Yin, M. A decomposition-based memetic algorithm using helper objectives for shortwave radio broadcast resource allocation problem in china. *Appl. Soft Comput.* **2020**, *91*, 106251. [CrossRef]
39. Jiang, E.; Wang, L.; Lu, J. Modified multi-objective evolutionary algorithm based on decomposition for low-carbon scheduling of distributed permutation flow-shop. In Proceedings of the 2017 IEEE Symposium Series on Computational Intelligence, Honolulu, HI, USA, 27 November–1 December 2017; pp. 1–7.
40. Abedi, M.; Chiong, R.; Noman, N.; Zhang, R. A multi-population, multi-objective memetic algorithm for energy-efficient job-shop scheduling with deteriorating machines. *Expert Syst. Appl.* **2020**, *157*, 113348. [CrossRef]
41. Kurdi, M. An improved island model memetic algorithm with a new cooperation phase for multi-objective job shop scheduling problem. *Comput. Ind. Eng.* **2017**, *111*, 183–201. [CrossRef]
42. Minguillon, F.E.; Stricker, N. Robust predictive–reactive scheduling and its effect on machine disturbance mitigation. *CIRP Ann.* **2020**, *69*, 401–404. [CrossRef]
43. Deng, J.; Wang, L.; Wu, C.; Wang, J.; Zheng, X. A competitive memetic algorithm for carbon-efficient scheduling of distributed flow-shop. In *International Conference on Intelligent Computing Lanzhou, China, 2–5 August 2016*; Springer: Cham, Switzerland, 2016; pp. 476–488.
44. Liu, X.; Zhan, Z.; Lin, Y.; Chen, W.; Gong, Y.; Gu, T.; Yuan, H.; Zhang, J. Historical and heuristic-based adaptive differential evolution. *IEEE Trans. Syst. Man Cybern. Syst.* **2018**, *49*, 2623–2635. [CrossRef]
45. Boejko, W.; Grabowski, J.; Wodecki, M. Block approach-tabu search algorithm for single machine total weighted tardiness problem. *Comput. Ind. Eng.* **2006**, *50*, 1–14. [CrossRef]
46. Osman, I.; Potts, C. Simulated annealing for permutation flow-shop scheduling. *Omega* **1989**, *17*, 551–557. [CrossRef]
47. Duan, J.; Sun, W.; Li, J.; Xu, Y. A Flowshop rescheduling algorithm based on migrating birds optimization. *Control. Eng. China* **2017**, *24*, 1656–1661. (In Chinese)