mathematics

MDPI

Article

A Self-Adaptive Memetic Algorithm for Distributed Job Shop Scheduling Problem

Guangchen Wang, Peng Wang * and Honggang Zhang

College of Weaponry Engineering, Naval University of Engineering, Wuhan 430033, China; 2120203075@nue.edu.cn (G.W.); 0912061002@nue.edu.cn (H.Z.)
* Correspondence: 0912101003@nue.edu.cn

Abstract: Distributed scheduling has become a common manufacturing mode, and the distributed job scheduling problem (DJSP) has attracted more manufacturers and researchers in the field of operation research. For the distributed scheduling problem, it emphasizes the flexibility of factory assignment and determines the sequence of operation related to each machine in related factories. In this paper, a mixed-integer linear programming model for the DJSP is formulated to be optimized by an SMA. Also in this paper, a self-adaptive memetic algorithm (SMA) is proposed to obtain a near-optimal solution in a limited time for the DJSP. To strengthen the effectiveness of the SMA, an independent encoding is designed with jobs assigned to factories and the sequence of operation. In the proposed algorithm, various local search strategies related to the critical path in the critical factory are designed to enhance the quality of the solution. Moreover, the self-adaptive scheme for solution improvement is formulated to reduce the search time and avoid prematurity effectively. To demonstrate the performance of the proposed algorithm, numerical experiments are carried out on 120 different instances extended from the well-known job shop scheduling benchmarks. The proposed SMA has updated 30 instance records in 120 instances and it has obtained the 91 best records in 120 instances. According to the comparison, an SMA is a more effective algorithm that could update several records of benchmarks.

Keywords: distributed job shop scheduling problem; self-adaptive memetic algorithm; makespan; chromosome representation

MSC: 90B80

Citation: Wang, G.; Wang, P.; Zhang, H. A Self-Adaptive Memetic Algorithm for Distributed Job Shop Scheduling Problem. *Mathematics* **2024**, *12*, 683. https://doi.org/10.3390/math12050683

Academic Editor: Frank Werner

Received: 4 December 2023
Revised: 30 January 2024
Accepted: 2 February 2024
Published: 26 February 2024

1. Introduction

Due to the nature of global economics and the quick reaction of the market, classical manufacturing does not satisfy the needs of the development. Distributed production and scheduling have become the new trend with their rapid reaction and low cost [1]. Thus, distributed scheduling has become a common manufacturing mode in the manufacturing industry, and the distributed scheduling problem has attracted more researchers in scheduling optimization. The distributed scheduling problem deals with the assignment of jobs to various factories with different geographical locations and by searching the sequence of operations on each machine [2].

The job shop scheduling problem (JSP) is a kind of classical complex scheduling problem among scheduling optimization research [3]. It has been proven as an NP-hard and one of the most intractable combinatorial problems [4]. A feasible solution to the JSP should satisfy the order of operation related to the machine. As the operations of different jobs could be arranged independently, there are a maximum of $(n!)^m$ solutions with n jobs with m machines [5]. Due to the increasing size of the problem, the explosive exponential growth in the number of candidate solutions will increase the difficulty of searching for optimal solutions. According to the predetermined sequence of operations, each operation has been allocated to the required machine with a fixed processing time. To solve the JSP, there have

been numerous studies related to various methods for solving the JSP in the last six decades. Disjunctive graph representation and enumerative methods based on the disjunctive graph were proposed to apply to the JSP. As more methods were proposed to solve the JSP, they could be classified into an exact method, heuristic and meta-heuristic. Branch and bound [6] is one of the exact methods that could search for the optimal solution with exponential computational complexity. However, the computational time will grow exponentially when the size of the problem grows. When solving the JSP on a large scale, heuristic and meta-heuristic methods are competitive, offering relatively good solutions within an acceptable time. Other advantages of the approximate method include dispatching rules, heuristic bottlenecks and neural networks [7], as well as constraint satisfaction [8]. Meta-heuristics with various local searchers have been developed for the JSP, such as genetic algorithm (GA) [9], simulated annealing (SA) [10,11], tabu search (TS) [12–14], ant optimization algorithm [15], memetic algorithm [16] and particle swarm optimization (PSO) [17]. Various algorithms could be potentially used for effective application in job scheduling, such as a snake optimizer [18] and smell agent optimization [19].

Compared with the JSP, the DJSP has an additional sub-problem with assignment for each job to various factories. Thus, the DJSP has many more difficulties than the JSP as an NP-hard problem [20]. It could be regarded as the JSP when the number of factories is set as one. In fact, the DJSP is different from the flexible job shop problem (FJSP). In the FJSP, all operations are in the same workshop and each operation has different available machines. But in the DJSP, all operations of the same job should be assigned to the same factory, yet each job could select any factory. When solving the DJSP, meta-heuristics are regarded as an effective method that is of computational capacity. Chaouch et al. [21] proposed an ant colony optimization algorithm (ACO) for the DJSP to optimize the makespan. Hsu et al. [22] developed the agent-based fuzzy constraint-directed negotiation mechanism (AFCN) with different negotiation strategies, such as competitive, win–win and collaborative strategies for the DJSP. As for the energy-efficient DJSP, Jiang et al. [23] presented a modified multi-objective evolutionary algorithm with decomposition (MMOEA/D), with a collaborative search and three problem-specific local intensification heuristics to optimize the makespan and energy consumption simultaneously. Şahman [24] proposed a discrete spotted hyena optimizer (DSHO) for the DJSP.

In this paper, a mathematical mixed-integer programming for the DJSP and an effective self-adaptive memetic algorithm (SMA) with various update operators and enhanced strategies are proposed to optimize the makespan for the DJSP. Various meta-heuristics [23,25,26] have been applied to various scheduling problems, and these have shown a satisfactory performance for distributed scheduling and job shop scheduling problems. In this study, a self-adaptive algorithm is designed to solve the DJSP. In this algorithm, the generation changes while the evaluation times change, and new individuals replace the previous generation with low quality. Furthermore, different local search strategies are designed to enhance the quality of the solution. To validate the effectiveness of the proposed algorithm, 120 instances for the DJSP [27] have been extended from Taillard's instances [28] related to the JSP. Furthermore, the Taguchi method of design of experiments (DOE) is applied in parameter setting. Compared with the existing algorithms for the DJSP [27], the SMA has great outperformance and reaches the smallest makespan for most instances.

The rest of the paper is organized as follows. In Section 2, the problem statement of the DJSP is formulated. Then, the details of an SMA for the DJSP with makespan criterion are presented in Section 3. In Section 4, parameter setting and computational results are compared with other methods for the DJSP. Finally, the conclusion and suggestions for future research are presented in Section 5.

2. Problem Definition and Formulation

2.1. Problem Definition

The classical DJSP could be described as a set of $N = \{1, \ldots, n\}$ jobs that would be assigned to $F = \{1, \ldots, f\}$ identical factories with a set of $M = \{1, \ldots, m\}$ machines.

The objective of optimization is to search for an optimal scheduling solution that has a minimized measurement of time, such as tardiness and makespan. The most common criteria of the existing literature is to optimize the makespan which is the maximum completion time among all factories. DJSP could be regarded as two sub-problems: assign jobs to geographically distributed factories and range the suitable scheduling operation sequence of each machine while optimizing a predefined objective.

Each job has a set of operations related to different machines and the sequence of operations has a different predetermined order. Therefore, different scheduling sequences will result in different values of criteria. All jobs should be assigned to various factories and any factory cannot be empty. Once one job is assigned to a factory, all operations related to the job should be processed in this factory, and it cannot be moved to another factory until its operations are finished. All machines and jobs are available at the start time, and there is no precedence constraint corresponding to operations of different jobs. One machine can only operate one job at the same time and one job can be performed by only one machine at the same time. The processing time of each operation is given with a specific machine and each operation has no specified release time or due time. The preparation time and transportation time are both negligible in this study.

To illustrate the DJSP clearly, there is a small size example with five jobs, two identical factories and two machines. Each job has two related operations, and each job is assigned two identical factories. The related processing time and constraint of the route of operations are shown in Table 1. For example, operation 1 of job 1 should be performed on machine 2 for 1 min. Then, operation 2 of job 1 should be performed on machine 1 for 3 min. It is scheduled as a Gantt chart of two factories, as shown in Figure 1. Job 1 and job 4 are assigned to factory 1, and job 2, job 3 and job 5 are assigned to factory 2. The completion time of factory 1 is 8 and the completion time of Factory 2 is 7. Therefore, the makespan is 8.

Table 1. The processing time of operation.

Job Index	Machine		Operation Route
	1	2	
1	3	1	{2, 1}
2	2	1	{1, 2}
3	2	2	{1, 2}
4	2	3	{2, 1}
5	3	1	{2, 1}

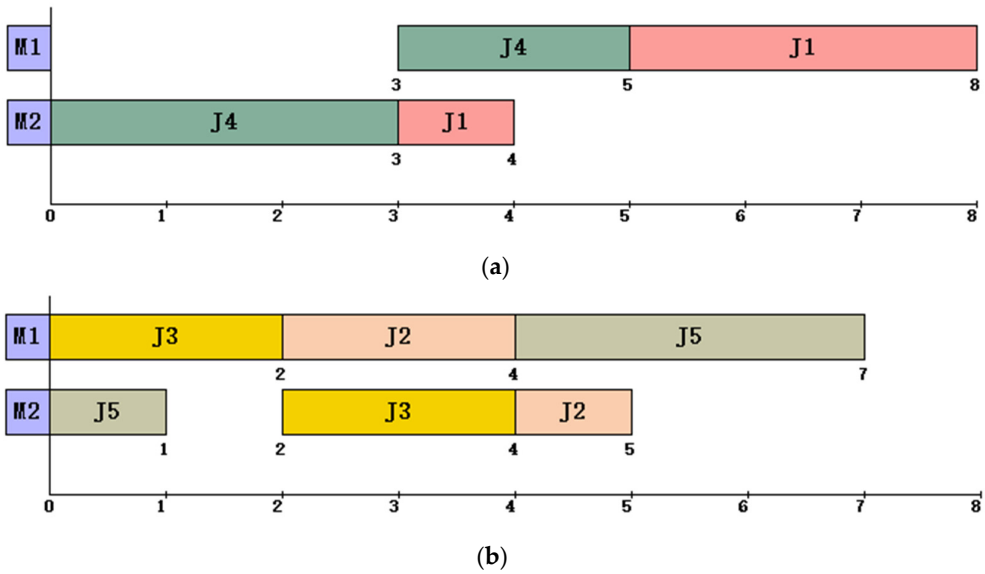

Figure 1. Gantt chart of instance. (**a**) Gantt chart of Factory 1; (**b**) Gantt chart of Factory 2.

2.2. Problem Formulation

Job data:

n : the number of jobs; $j, h \in \{0, 1, 2, \ldots, n\}$, where job 0 represents the dummy value to assist in defining the first one processed on a machine.

Processing environment:

m : the number of machines; $i, l \in \{1, 2, \ldots, m\}$

f : the number of distributed factories; $k = \{1, 2, \ldots, f\}$

$X_{h,j,i,k}$: a binary value, which equals 1 when job j is operated on machine i after job h in distributed factory k

$a_{j,l,i}$: a binary value, which equals 1 when job j is operated on the machine i after the machine l in the process route and 0 otherwise.

$S_{j,i}$: the start time of job j operated on machine i

$p_{j,i}$: the processing time related to job j on a machine i

Optimality criteria:

A: a large positive number

$$\text{Minimize} C_{\max} \tag{1}$$

which is subject to:

$$\sum_{h=0, h\neq j}^{n} \sum_{k=1}^{f} X_{h,j,1,k} = 1 \ \ \forall j \tag{2}$$

$$\sum_{h=0, h\neq j}^{n} X_{h,j,i,k} = \sum_{h=0, h\neq j}^{n} X_{h,j,1,k} \ \ \forall j, i > 1, \ k \tag{3}$$

$$\sum_{j=1, j\neq h}^{n} X_{h,j,i,k} = \sum_{j=0 \ j\neq h}^{n} X_{j,h,1,k} \ \ \forall h > 0, i, \ k \tag{4}$$

$$\sum_{j=1}^{n} X_{0,j,i,k} = 1 \ \forall i, k \tag{5}$$

$$\sum_{h=1}^{n} \left(X_{h,j,i,k} + X_{j,h,i,k} \right) \leq 1 \ \ \forall j < n, h > j, i \tag{6}$$

$$S_{j,i} \geq S_{j,l} + p_{j,l} \ \ \forall j, i, l \neq i \Big| a_{j,l,i} = 1 \tag{7}$$

$$S_{j,i} \geq S_{h,i} + p_{h,i} - A\left(1 - X_{h,j,i,k}\right) \ \forall h < 0, j \neq h, i, k \tag{8}$$

$$C_{\max} = \max\{S_{j,i} + p_{j,i}\} \ \forall j, i \tag{9}$$

$$S_{j,i} \geq 0 \ \forall j, i \tag{10}$$

$$X_{h,j,i,k} = \{0, \ 1\} \forall j, h \neq j, i, k \tag{11}$$

Equation (1) represents the objective function as the makespan of DJSP. Constraint (2) means each job has and can only be assigned to one factory. Constraints (3) and (4) mean all operations of one job should be assigned to the determined factory. Constraint (5) determines the first job operated on each machine. Constraint (6) means each operation of the job has only an adjacent operation. Constraint (7) determines the start time of former and latter machine related to the same job. Constraint (8) determines the start time of former and latter job related to the same machine. Constraint (9) represents the calculation of the makespan. Constraint (9) and (10) denotes the decision variables.

3. Self-Adaptive Memetic Algorithm for DJSP

The memetic algorithm was proposed as a population-based evolution algorithm which is similar to GA. In recent years, it has evolved as an algorithm frame and it has flexibility that could be adjusted for various scheduling problems [16]. In light of the

advantages of the memetic algorithm in terms of global and local search ability, it is suitable to be utilized with the DJSP on problems with great difficulty and with a large solution space and wide search space. In the study, SMA is designed with self-adaptive update operators and local search strategies for DJSP. In self-adaptive update procedure, different update operators could be selected with different evolution standards. E is a parameter to choose the update operators and it is influenced by evaluation times and evolution degree. E is determined by initial energy E_0, search energy E_1 and evolution degree. E_0 is initial energy and it is a random value in the range $[-1, 1]$. E_1 is determined by current the unchanged count and max unchanged count. Once the best solution has not been updated for a long time, the latter evolution will tend toward a local search with great possibility. The update operators are crossover and mutation. The details of SMA are shown in Algorithm 1.

Algorithm 1: Pseudo code of SMA

While the stopping condition ($unchangecount < maxunchangecount$ or $t < T$) is not satisfied **do**
  Evaluate the fitness of each individual
    For each individual (X_i) do
      Update the initial energy $E_0 = 2rand() - 1$ and search energy $E_1 = 1 - \frac{C}{4 \times M}$
      Update the $E = 2E_0E_1\left(1 - \frac{t}{T}\right)$. M denotes the max unchanged count, C denotes current the unchanged count, T denotes the max evaluation times and t denotes the current evaluation times.
      If $|E| \geq 1$ (exploration)
        If $q \geq 0.5$, select the best individual and apply the binary selection operator to select the individual in the population to go through the update operators. $t \leftarrow t + 1$
        If $q \geq 0.5$, select the best individual and second-place individual to go through the update operators. $t \leftarrow t + 1$
      If $|E| < 1$ (exploitation)
        If (r ≥ 0.5&|E| ≥ 0.5) then
          Select the best individual and second place individual to go through the update operators and insert local search. $t \leftarrow t + 1$
        If ($r \geq 0.5$&$|E| < 0.5$) then
          Update the last $1/4$ individuals with the newly generated individuals and sort the new population. Select the best individual and random individual to go through the update operators and insert local search. $t \leftarrow t + 1$
        If ($r < 0.5$&$|E| \geq 0.5$)
          Select the best individual and random individual to go through the update operators and local search based factory assignment. $t \leftarrow t + 1$
        Else if (r < 0.5&|E| < 0.5) then
          Update the last $1/4$ individuals with the newly generated individuals and sort the new population. Select the best individual and random individual to go through the update operators and local search based factory assignment. $t \leftarrow t + 1$
      If the best solution of the population is not updated, $C \leftarrow C + 1$
      Else $C \leftarrow 0$
    End for
End while

3.1. Solution Representation and Initialization

In this study, a chromosome representation with factory assignment and operation-based encoding schemes is introduced to SMA. For the first level, there is a job-to-cell assignment, which is an independent chromosome to denote the factory for DJSP. In population initialization, $1/5$ of the population of individuals are initialized with the load-balance heuristic rules [24] to enhance the quality of the population. The other population individuals are initialized with a random method to ensure the diversity of the population. The length of the first level is related to the index of the job. Each gene represents the factory assignment of each job. Regarding the operation-based encoding scheme, it is widely applied in job shop scheduling problems proposed by Bierwirth [29]. There is a

small instance in Figure 2 relating to five jobs with 10 operations which are assigned to two factories, which could be decoded as shown in Figure 1. In the job assignment, it has five genes, which means each job has its related factory, such as job 1 being assigned to factory 1. In an operation sequence, its length is related to the number of operations. The first gene is "5", and it means the first operation of job 5 is the first one to be performed.

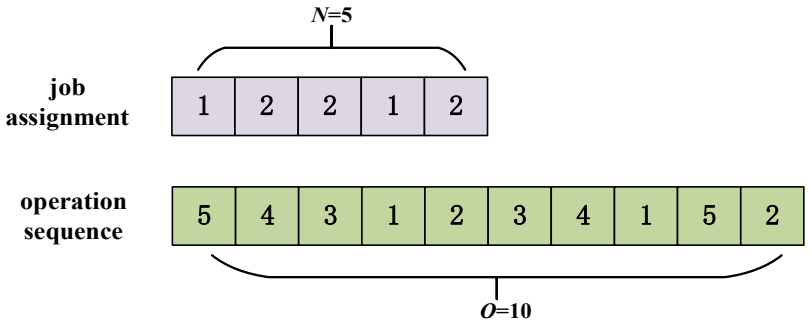

Figure 2. Chromosome of the instance.

The first step of decoding is to assign each job to its related factory. Then, permutate the operation according to the operation sequence. Next, there are two jobs with four operations {4, 1, 4, 1} in factory 1. The other three jobs with six operations {5, 3, 2, 3, 5, 2} are assigned in factory 2.

3.2. Update Operators

Genetic operators are widely applied in the existing literature [30] in an exploitation and exploration manner. In this study, genetic operators are applied as update operators which are designed for the chromosome representation.

Crossover operators could enhance the diversity of the solutions and offer a different crossover. But for discrete problems with solution constraints, it may result in an unfeasible solution. Thus, two different crossover operators exist for different levels of chromosomes. For chromosomes with factory assignment, binomial crossover [31] is applied to avoid an unfeasible solution in update operators. As shown in Figure 3, each gene has the same range from 1 to the max value of the factory. Therefore, the factory assignment of offspring would be distributed within a limited range. When the value of possibility is no less than 0.5, the gene in parent 1 is retained in offspring 1; otherwise, the gene is retained in offspring 2. When the value of possibility is less than 0.5, the gene in parent 1 is retained in offspring 2; otherwise, the gene is retained in offspring 1. But after the crossover operator in the chromosome of factory assignment, it is vital to ensure that each factory has at least one job. In case there is an empty factory, one job would be selected randomly from the factory which possesses at least two jobs, and the job would be assigned to the empty factory.

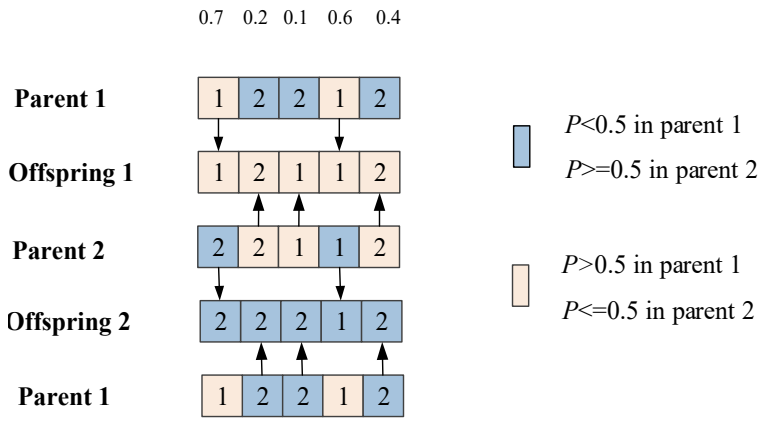

Figure 3. Binomial crossover for the chromosome of factory assignment.

As the crossover of the operation sequence, a two-point mapping crossover is applied as an update operator. As shown in Figure 4, two points are selected randomly, and two fragments of parent individuals are exchanged. The fragments behind the left selected point in parent 1 as {5, 4, 3} would be retained in offspring 1. This is the same procedure from parent 2 to offspring 2 as {1, 5, 2}. The fragment in parent 1 between two selected points as {1, 2, 3, 4} would be matched with a sequence in parent 2 as {2, 1, 3, 4}. The matched sequence will be retained in offspring 1 as {2, 1, 3, 4}. The same procedure is performed in offspring 2 from {4, 1, 2, 5} to {4, 1, 5, 2}.

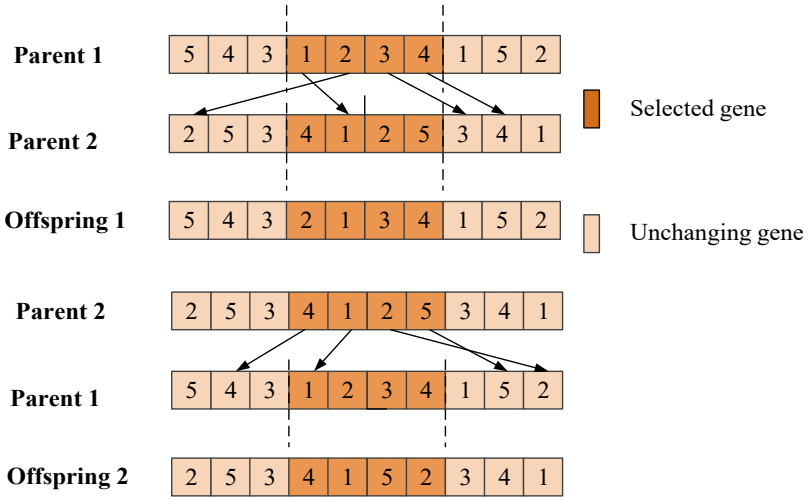

Figure 4. Two-point mapping crossover for the chromosome of operation sequence.

Differently from the crossover operator, the mutation operator is normally applied with a low possibility. As the factory assignment is at the first level, a single-point reset is applied in the mutation operator. In this procedure, as shown in Figure 5a, one point is selected randomly and its gene is changed within a limited range. As in Figure 5a, the third gene "2" would be transformed to "1", which means job 3 would be removed from factory 2 and inserted into factory 1.

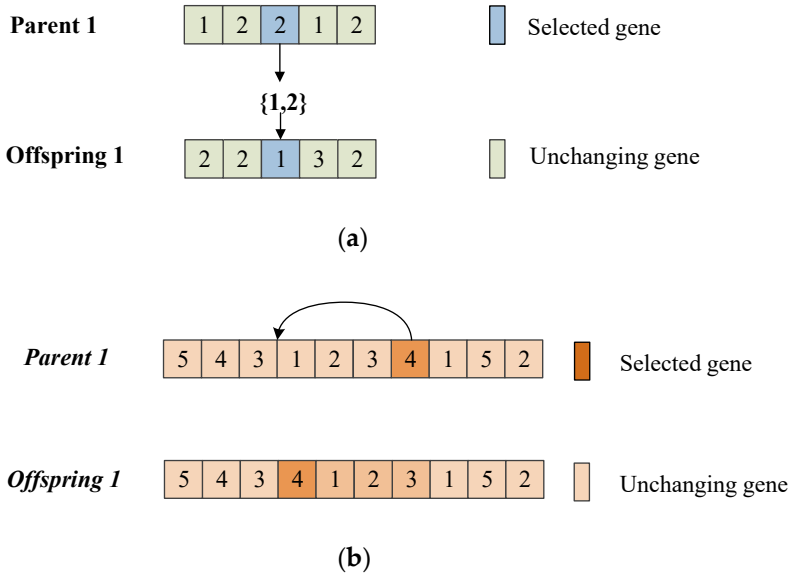

Figure 5. Mutation operators for the factory assignment and operation sequence. (**a**) Single-point reset for the chromosome of factory assignment; (**b**) Single-point insert for the chromosome of operation sequence.

As for the operation sequence, to avoid an unfeasible solution, a single-point insert is applied in the mutation procedure. As shown in Figure 5b, two different points are selected randomly. Then, insert the selected job to the selected position. In this procedure, the feasibility of the updated solution can be ensured.

3.3. Local Search

For distributed scheduling problems with the optimization of the makespan, the factory with the max value of completion time is named the critical factory [32]. The value of the makespan is equal to the completion time of the critical factory. In this section, local search is applied to enhance the exploitation of the proposed method. The local search based on factory assignment and operation in the critical factory is designed to optimize the completion of the critical factory.

Local search based on factory assignment is proposed to adjust the factory assignment to reduce the completion time of the critical factory. As shown in Algorithm 2, the first step is to calculate the completion time of each factory and confirm the critical factory. Then, enter the loop for each job in the critical factory and exchange its possible factory except for the critical factory. Next, evaluate each candidate solution and compare it to the existing value of makespan. In the end, update the optimal solution and value of the makespan as an input into the next loop.

Algorithm 2: Local search based on factory assignment

```
Calculate the completion time of each factory and makespan C_max
  denote the factory with max completion time as f_c
    for each job j in f_c
      for each job in other factory as f_o except f_c
        exchange their factory assignment f_c and f_o
        evaluate the new makespan as C'_max
          if C'_max < C_max
            update the factory assignment and C_max
          end if
        end for
end for
```

A local search based on the operation in the critical factory is aimed at adjusting the operation in the critical factory to optimize the makespan. The details of the procedure are shown in Algorithm 3. The first step is to identify the critical factory. Then, select the operation in the critical factory randomly. Next, try to insert the selected operation in a possible position. Finally, evaluate the new solution as the new makespan and update the new solution with the minimum value of makespan.

Algorithm 3: Local search based on operation in critical factory

```
Calculated the completion time of each factory and makespan C_max
  denote the factory with max completion time as f_c
    select the operation O_i randomly in f_c
      for each possible position p in the operation sequence
        for k = 1 to f
            insert O_i to position p and reassign it to factory k
            evaluate the new makespan as C'_max
            if C'_max < C_max
              update the factory assignment, operation sequence and C_max
            end if
          end for
      end for
```

4. Experiment Evaluation

4.1. Parameter Setting

To verify the effectiveness of the proposed method, SMA is compared with other existed methods for DJSP. The existing test instances for DJSP, ACO [33], HACO [27], DAHACO [27], DPSO [24] and DSHO [24] have been applied with and have their computational records in existing studies [25]. These experiments were conducted on well-known benchmarks with various numbers of factories [28]. In this study, all experiments are conducted on a computer with an AMD Ryzen 7 5800H (TSMC, Taiwan, China) processor with Radeon Graphics 3.20 GHz 16 GB RAM.

The SMA contains five main parameters: the stop condition is related to population size, the max unchanged count and evaluation time. To verify the effect of the parameter and find the optimal level of the parameter, the Taguchi method [34] is the parameter setting technique of experiments. The level of the 5 main parameters is shown in Table 2, and each parameter has four levels of various values. To avoid a huge number of unnecessary experiments as 5^4 experiment settings, the DOE could reduce this number to 16 experiments with five replications. According to an existing experiment [27] on DJSP, these different levels of parameters are applied in Tai01 (15 jobs, 15 machines). Orthogonal arrays are applied to evaluate the effect of factors. As shown in Table 3, each row represents an experiment with a combination of parameter levels. The orthogonal array $L_{16}(4^5)$ could be regarded as 16 parameter combinations, and each combination would be repeated five times independently.

Table 2. Level of main parameters setting.

Level	1	2	3	4
N	20	50	100	200
M	50	100	150	200
T	50,000	100,000	150,000	200,000
α	0.8	0.85	0.9	0.95
β	0.05	0.1	0.15	0.2

Table 3. The mean the of S/N Ratio for different parameter settings.

No	N	M	T	α	β	*RPD* (Mean)	*S/N* Ratio
1	20	50	50,000	0.80	0.05	29.3	−29.34
2	20	100	100,000	0.85	0.10	28.6	−29.13
3	20	150	150,000	0.90	0.15	27.5	−28.79
4	20	200	200,000	0.95	0.20	26.7	−28.53
5	50	50	100,000	0.90	0.20	25.7	−28.20
6	50	100	50,000	0.95	0.15	29.4	−29.37
7	50	150	200,000	0.80	0.10	25.8	−28.23
8	50	200	150,000	0.85	0.05	24.9	−27.92
9	100	50	150,000	0.95	0.10	22.4	−27.00
10	100	100	200,000	0.90	0.05	20.7	−26.32
11	100	150	50,000	0.85	0.20	24.1	−27.64
12	100	200	100,000	0.80	0.15	21.9	−26.81
13	200	50	200,000	0.85	0.15	24.7	−27.85
14	200	100	150,000	0.80	0.20	26.1	−28.33
15	200	150	100,000	0.95	0.05	27.8	−28.88
16	200	200	50,000	0.90	0.10	25.6	−28.16

To evaluate the effect of parameter factor with the standard evaluation metric, a signal-to-noise ratio (S/N ratio) [27] is introduced to evaluate the impact of noise factor on algorithm performance. The S/N ratio was proposed for the ratio of signal to noise in an electronic device or system, and its large value means a higher level of robustness against noise. The unit of the S/N ratio is named as decibel (dB). With regard to the difference in

the performance characteristics, the reflected performance of the S/N is different. They could be classified as follows: the nominal value is considered better, the smaller value is considered better (STB) and the larger value is considered better. As the objective of the instance is the makespan, it could be classified that the smaller value of the S/N ratio is better.

To define the performance of the various algorithms, Relative Percentage Deviation (RPD) [27] is applied in this study, and it could be defined as follows:

$$RPD = \frac{Alg - Min}{Min} \times 100 \tag{12}$$

where *Alg* is denoted as the makespan obtained by the corresponding algorithm with a related instance and *Min* is the lowest value of the makespan among all algorithms for the same instance. As the objective is to minimize the *S/N ratio* and the necessity of standardizing the *S/N ratio* with the different instances, the *S/N ratio* could be calculated as follows:

$$S/N\ ratio = -10 \times \log_{10}(RPD)^2 \tag{13}$$

Thus, the *S/N ratio* is set as a reflected indicator to determine the level of the SMA in this study in the orthogonal array. The lower value of the *S/N ratio* is more desirable in the parameter setting.

From Table 3 and Figure 6, the response of different parameter settings is the average value of the *S/N ratio*, and its value is calculated based on Equations (12) and (13). Thus, the parameter values are set as follows: N = 100, M = 200, T = 200,000, α = 0.9, β = 0.10.

Figure 6. The trend of the factor level.

4.2. Comparison of SMA with Other Algorithms

To verify the effectiveness of the proposed method, SMA is compared with ACO [33], HACO [27], DAHACO [27], DPSO [24], and DSHO [24] with instances in an existing study [27] for DJSP. It has 120 instances, with 20 different instances with different numbers of jobs and machines, and 6 different kinds of factories (f = 2, 3, 4, 5, 6, 7), the value of the makespan is presented in Tables 4–9.

Table 4. Comparison with different algorithms with 2 factories.

Problem	Size	DAHACO	HACO	ACO	DPSO	DSHO	SMA
Tai01	15 × 15	1381	1433	1708	1156	1067	**1047**
Tai02	15 × 15	1213	1195	1570	1138	**1076**	**1076**
Tai03	15 × 15	1376	1484	1616	1178	**1103**	**1103**

Table 4. *Cont.*

Problem	Size	DAHACO	HACO	ACO	DPSO	DSHO	SMA
Tai04	15 × 15	1246	1406	1740	1122	**1058**	1080
Tai05	15 × 15	1380	1505	2188	1278	**1135**	**1135**
Tai06	15 × 15	1347	1467	1627	1147	**1070**	1108
Tai07	15 × 15	1402	1561	1811	1183	1102	**1098**
Tai08	15 × 15	1295	1494	1713	1141	**1082**	**1082**
Tai09	15 × 15	1434	1445	1947	1232	**1167**	1178
Tai10	15 × 15	1259	1594	1886	1094	**1080**	**1080**
Tai11	20 × 15	1444	1641	1976	1323	**1211**	**1211**
Tai12	20 × 15	1483	1696	2117	1331	**1236**	1292
Tai13	20 × 15	1511	1647	2253	1349	**1173**	**1173**
Tai14	20 × 15	1450	1477	2207	1192	**1109**	**1109**
Tai15	20 × 15	1571	1852	2138	1288	**1138**	**1138**
Tai16	20 × 15	1564	1535	2257	1254	**1219**	**1219**
Tai17	20 × 15	1618	1681	2174	1359	**1249**	1311
Tai18	20 × 15	1505	1664	2209	1315	**1188**	**1188**
Tai19	20 × 15	1555	1604	2157	1156	1067	**1047**
Tai20	20 × 15	1543	1759	2108	1138	**1076**	**1076**

The minimum value of the makespan is in bold font.

Table 5. Comparison with different algorithms with 3 factories.

Problem	Size	DAHACO	HACO	ACO	DPSO	DSHO	SMA
Tai01	15 × 15	1381	1433	1708	1075	**1002**	**1002**
Tai02	15 × 15	1213	1195	1570	1012	**969**	**969**
Tai03	15 × 15	1376	1484	1616	1020	**986**	**954**
Tai04	15 × 15	1246	1406	1740	1006	965	**961**
Tai05	15 × 15	1380	1505	2188	1025	**1003**	**1003**
Tai06	15 × 15	1347	1467	1627	1028	967	**943**
Tai07	15 × 15	1402	1561	1811	1083	1024	**1017**
Tai08	15 × 15	1295	1494	1713	1001	989	**977**
Tai09	15 × 15	1434	1445	1947	1118	**1002**	1025
Tai10	15 × 15	1259	1594	1886	1024	**977**	1005
Tai11	20 × 15	1444	1641	1976	1138	**1067**	**1067**
Tai12	20 × 15	1483	1696	2117	1189	**1059**	1117
Tai13	20 × 15	1511	1647	2253	1055	**995**	**995**
Tai14	20 × 15	1450	1477	2207	1082	**1056**	**1056**
Tai15	20 × 15	1571	1852	2138	1093	**1000**	1096
Tai16	20 × 15	1564	1535	2257	1153	1127	**1106**
Tai17	20 × 15	1618	1681	2174	1233	**1111**	1160
Tai18	20 × 15	1505	1664	2209	1209	1121	**1115**
Tai19	20 × 15	1555	1604	2157	1100	**1038**	**1038**
Tai20	20 × 15	1543	1759	2108	1118	**1077**	1102

The minimum value of the makespan is in bold font.

Table 6. Comparison with different algorithms with 4 factories.

Problem	Size	DAHACO	HACO	ACO	DPSO	DSHO	SMA
Tai01	15 × 15	1381	1433	1708	983	974	**963**
Tai02	15 × 15	1213	1195	1570	986	**942**	**942**
Tai03	15 × 15	1376	1484	1616	941	**922**	**922**
Tai04	15 × 15	1246	1406	1740	**921**	**921**	**921**
Tai05	15 × 15	1380	1505	2188	970	**940**	**940**
Tai06	15 × 15	1347	1467	1627	986	964	**944**
Tai07	15 × 15	1402	1561	1811	975	942	**935**
Tai08	15 × 15	1295	1494	1713	964	**963**	**963**
Tai09	15 × 15	1434	1445	1947	983	**982**	**982**
Tai10	15 × 15	1259	1594	1886	**944**	**944**	**944**

Table 6. *Cont.*

Problem	Size	DAHACO	HACO	ACO	DPSO	DSHO	SMA
Tai11	20 × 15	1444	1641	1976	1069	1002	**988**
Tai12	20 × 15	1483	1696	2117	1096	1056	**1041**
Tai13	20 × 15	1511	1647	2253	1016	**955**	974
Tai14	20 × 15	1450	1477	2207	994	**990**	**990**
Tai15	20 × 15	1571	1852	2138	1048	**970**	988
Tai16	20 × 15	1564	1535	2257	1050	**986**	1036
Tai17	20 × 15	1618	1681	2174	1101	**1025**	1029
Tai18	20 × 15	1505	1664	2209	1064	**988**	1020
Tai19	20 × 15	1555	1604	2157	997	**946**	995
Tai20	20 × 15	1543	1759	2108	1025	**976**	1022

The minimum value of the makespan is in bold font.

Table 7. Comparison with different algorithms with 5 factories.

Problem	Size	DAHACO	HACO	ACO	DPSO	DSHO	SMA
Tai01	15 × 15	1381	1433	1708	**963**	**963**	**963**
Tai02	15 × 15	1213	1195	1570	949	949	**942**
Tai03	15 × 15	1376	1484	1616	978	978	**933**
Tai04	15 × 15	1246	1406	1740	921	921	**917**
Tai05	15 × 15	1380	1505	2188	**940**	**940**	**940**
Tai06	15 × 15	1347	1467	1627	937	907	**890**
Tai07	15 × 15	1402	1561	1811	978	978	**953**
Tai08	15 × 15	1295	1494	1713	**963**	**963**	**963**
Tai09	15 × 15	1434	1445	1947	**982**	**982**	**982**
Tai10	15 × 15	1259	1594	1886	921	**906**	909
Tai11	20 × 15	1444	1641	1976	**949**	**949**	954
Tai12	20 × 15	1483	1696	2117	1037	**1012**	**1012**
Tai13	20 × 15	1511	1647	2253	980	960	**951**
Tai14	20 × 15	1450	1477	2207	**990**	**990**	**990**
Tai15	20 × 15	1571	1852	2138	955	**912**	937
Tai16	20 × 15	1564	1535	2257	990	**974**	999
Tai17	20 × 15	1618	1681	2174	1020	**979**	**979**
Tai18	20 × 15	1505	1664	2209	969	**931**	933
Tai19	20 × 15	1555	1604	2157	974	**935**	**935**
Tai20	20 × 15	1543	1759	2108	945	**928**	957

The minimum value of the makespan is in bold font.

Table 8. Comparison with different algorithms with 6 factories.

Problem	Size	DAHACO	HACO	ACO	DPSO	DSHO	SMA
Tai01	15 × 15	1381	1433	1708	**963**	**963**	**963**
Tai02	15 × 15	1213	1195	1570	**942**	**942**	**942**
Tai03	15 × 15	1376	1484	1616	933	933	**921**
Tai04	15 × 15	1246	1406	1740	921	921	**911**
Tai05	15 × 15	1380	1505	2188	**940**	**940**	**940**
Tai06	15 × 15	1347	1467	1627	903	877	**873**
Tai07	15 × 15	1402	1561	1811	**935**	**935**	**935**
Tai08	15 × 15	1295	1494	1713	**963**	**963**	**963**
Tai09	15 × 15	1434	1445	1947	**982**	**982**	**982**
Tai10	15 × 15	1259	1594	1886	947	947	**896**
Tai11	20 × 15	1444	1641	1976	967	**949**	**949**
Tai12	20 × 15	1483	1696	2117	**1012**	**1012**	**1012**
Tai13	20 × 15	1511	1647	2253	923	923	**919**
Tai14	20 × 15	1450	1477	2207	**990**	**990**	**990**
Tai15	20 × 15	1571	1852	2138	943	**909**	913

Table 8. *Cont.*

Problem	Size	DAHACO	HACO	ACO	DPSO	DSHO	SMA
Tai16	20 × 15	1564	1535	2257	971	**936**	942
Tai17	20 × 15	1618	1681	2174	1010	986	**981**
Tai18	20 × 15	1505	1664	2209	967	**903**	917
Tai19	20 × 15	1555	1604	2157	934	**920**	**920**
Tai20	20 × 15	1543	1759	2108	**928**	**928**	**928**

The minimum value of the makespan is in bold font.

Table 9. Comparison with different algorithms with 7 factories.

Problem	Size	DAHACO	HACO	ACO	DPSO	DSHO	SMA
Tai01	15 × 15	1381	1433	1708	**963**	**963**	**963**
Tai02	15 × 15	1213	1195	1570	**942**	**942**	**942**
Tai03	15 × 15	1376	1484	1616	933	933	**921**
Tai04	15 × 15	1246	1406	1740	921	921	**911**
Tai05	15 × 15	1380	1505	2188	**940**	**940**	**940**
Tai06	15 × 15	1347	1467	1627	899	875	**873**
Tai07	15 × 15	1402	1561	1811	**935**	**935**	**935**
Tai08	15 × 15	1295	1494	1713	**963**	**963**	**963**
Tai09	15 × 15	1434	1445	1947	**982**	**982**	**982**
Tai10	15 × 15	1259	1594	1886	944	944	**896**
Tai11	20 × 15	1444	1641	1976	956	**949**	**949**
Tai12	20 × 15	1483	1696	2117	**1012**	**1012**	**1012**
Tai13	20 × 15	1511	1647	2253	**919**	**919**	**919**
Tai14	20 × 15	1450	1477	2207	**990**	**990**	**990**
Tai15	20 × 15	1571	1852	2138	**909**	**909**	**909**
Tai16	20 × 15	1564	1535	2257	**932**	**932**	936
Tai17	20 × 15	1618	1681	2174	**979**	**979**	**979**
Tai18	20 × 15	1505	1664	2209	904	**900**	905
Tai19	20 × 15	1555	1604	2157	**920**	**920**	**920**
Tai20	20 × 15	1543	1759	2108	**934**	**934**	**934**

The minimum value of the makespan is in bold font.

From all results with different levels of factory, the value of the makespan is lower with a higher factory number value. It reflects the advantage of factory distribution and time saving in a distributed manufactory system. To manifest the comparison of different algorithms, the minimum value of the makespan is in bold font, which means the most competitive method in the corresponding instance. From the comparison result of the makespan, SMA has great performance in all instances in comparison with other compared methods.

To analyze the performance of the SMA, it obtains the 15 best results of the comparison and updates three instance records in 20 instances with two factories. It updates six instance records and obtains the 14 best results with 20 instances when it is related to three factories. When the number of factories is four, the proposed SMA obtains the 13 best results and updates five instance records with 20 instances. When the number of factories is five, it can obtain the 14 best instance records among all compared algorithms and it updates six instances record as the best records with 20 instances. When the number of factories is six, it obtains the 17 best records among all compared algorithms with 20 instances and it updates the six best instance records. It obtains the 18 best results of the comparison and updates four instance records in 20 instances with seven factories. To sum up, the proposed SMA has updated 30 instance records in 120 instances and it has obtains the 91 best records in 120 instances. In this comparison, it has a better performance when compared with the other algorithms.

5. Conclusions

In this study, a self-adaptive memetic algorithm is proposed to optimize the makespan of the distributed job shop scheduling problem and a mixed-integer linear programming model is formulated to describe the problem clearly. In the encoding procedure, concise encoding is designed with job assignment and operation sequence. In this way, job assignment and operation sequence could be adjusted independently. The effective initialization of the population with workload equilibrium mechanism and random generation could be balanced between the quality and diversity of the population. Secondly, the evolution procedure is adaptive with the knowledge of the existing population. In this way, different update operators could be selected according to the evaluation of individuals. Two kinds of local search are designed to improve the exploitation of the algorithm. In the experimental phase, 120 benchmark instances for the experiments are conducted in comparison with five existing algorithms. The proposed SMA has updated 30 instance records in 120 instances and it has obtained the 91 best records in 120 instances and it is improved as the effective algorithm for DJSP. Although this study is about SMA as an effective algorithm for the benchmarking of DJSP, SMA could be applied in real manufacturing cases in the future, and it could even be transformed into a multi-objective algorithm with conflicting objectives.

Author Contributions: Conceptualization, G.W. and P.W.; methodology, G.W.; validation, H.Z.; writing—original draft preparation, G.W.; writing—review, P.W.; supervision, P.W. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by National Natural Science Foundation of China (Grant No. 62101579).

Data Availability Statement: Data will be made available on request.

Acknowledgments: The authors are grateful to the responsible editor and the anonymous reviewers for their valuable comments and suggestions, which have greatly improved the earlier version of this paper.

Conflicts of Interest: The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

1. Jia, H.Z.; Fuh, J.Y.H.; Nee, A.Y.C.; Zhang, Y.F. Web-based multi-functional scheduling system for a distributed manufacturing environment. *Concurr. Eng. Res. Appl.* **2002**, *10*, 27–39. [CrossRef]
2. Chan, F.T.S.; Chung, S.H.; Chan, P.L.Y. An adaptive genetic algorithm with dominated genes for distributed scheduling problems. *Expert Syst. Appl.* **2005**, *29*, 364–371. [CrossRef]
3. Gao, L.; Li, X.; Wen, X.; Lu, C.; Wen, F. A hybrid algorithm based on a new neighborhood structure evaluation method for job shop scheduling problem. *Comput. Ind. Eng.* **2015**, *88*, 417–429. [CrossRef]
4. Garey, M.R.; Johnson, D.S.; Sethi, R. The complexity of flowshop and jobshop scheduling. *Math. Oper. Res.* **1976**, *1*, 117–129. [CrossRef]
5. Hoitomt, D.J.; Luh, P.B.; Pattipati, K.R. A practical approach to job-shop scheduling problems. *IEEE Trans. Robot. Autom.* **1993**, *9*, 1–13. [CrossRef]
6. Schaller, J.; Valente, J. Branch-and-bound algorithms for minimizing total earliness and tardiness in a two-machine permutation flow shop with unforced idle allowed. *Comput. Oper. Res.* **2019**, *109*, 1–11. [CrossRef]
7. Weckman, G.R.; Ganduri, C.V.; Koonce, D.A. A neural network job-shop scheduler. *J. Intell. Manuf.* **2008**, *19*, 191–201. [CrossRef]
8. Yuanyuan, T.; Shixin, L.; Dazhi, W. A constraint programming-based branch and bound algorithm for job shop problems. In Proceedings of the 2010 Chinese Control and Decision Conference, Xuzhou, China, 26–28 May 2010; pp. 173–178.
9. Watanabe, M.; Ida, K.; Gen, M. A genetic algorithm with modified crossover operator and search area adaptation for the job-shop scheduling problem. *Comput. Ind. Eng.* **2005**, *48*, 743–752. [CrossRef]
10. Suresh, R.; Mohanasundaram, K. Pareto archived simulated annealing for job shop scheduling with multiple objectives. *Int. J. Adv. Manuf. Technol.* **2006**, *29*, 184–196. [CrossRef]
11. Van Laarhoven, P.J.; Aarts, E.H.; Lenstra, J.K. Job shop scheduling by simulated annealing. *Oper. Res.* **1992**, *40*, 113–125. [CrossRef]
12. Chen, L.; Bostel, N.; Dejax, P.; Cai, J.; Xi, L. A tabu search algorithm for the integrated scheduling problem of container handling systems in a maritime terminal. *Eur. J. Oper. Res.* **2007**, *181*, 40–58. [CrossRef]
13. Eswaramurthy, V.; Tamilarasi, A. Hybridizing tabu search with ant colony optimization for solving job shop scheduling problems. *Int. J. Adv. Manuf. Technol.* **2009**, *40*, 1004–1015. [CrossRef]

14. Pezzella, F.; Merelli, E. A tabu search method guided by shifting bottleneck for the job shop scheduling problem. *Eur. J. Oper. Res.* **2000**, *120*, 297–310. [CrossRef]
15. Zhou, R.; Nee, A.; Lee, H. Performance of an ant colony optimisation algorithm in dynamic job shop scheduling problems. *Int. J. Prod. Res.* **2009**, *47*, 2903–2920. [CrossRef]
16. Gong, G.; Deng, Q.; Chiong, R.; Gong, X.; Huang, H. An effective memetic algorithm for multi-objective job-shop scheduling. *Knowl.-Based Syst.* **2019**, *182*, 104840. [CrossRef]
17. Lin, T.; Horng, S.; Kao, T.; Chen, Y.; Run, R.; Chen, R.; Lai, J.; Kuo, I. An efficient job-shop scheduling algorithm based on particle swarm optimization. *Expert Syst. Appl.* **2010**, *37*, 2629–2636. [CrossRef]
18. Hashim, F.A.; Hussien, A.G. Snake Optimizer: A novel meta-heuristic optimization algorithm. *Knowl.-Based Syst.* **2022**, *242*, 108320. [CrossRef]
19. Salawudeen, A.T.; Mu'azu, M.B.; Sha'aban, Y.A.; Adedokun, A.E. A Novel Smell Agent Optimization (SAO): An extensive CEC study and engineering application. *Knowl.-Based Syst.* **2021**, *232*, 107486. [CrossRef]
20. Naderi, B.; Azab, A. Modeling and heuristics for scheduling of distributed job shops. *Expert Syst. Appl.* **2014**, *41*, 7754–7763. [CrossRef]
21. Chaouch, I.; Driss, O.B.; Ghedira, K. A modified ant colony optimization algorithm for the distributed job shop scheduling problem. *Procedia Comput. Sci.* **2017**, *112*, 296–305. [CrossRef]
22. Hsu, C.; Kao, B.; Lai, K.R. Agent-based fuzzy constraint-directed negotiation mechanism for distributed job shop scheduling. *Eng. Appl. Artif. Intell.* **2016**, *53*, 140–154. [CrossRef]
23. Jiang, E.; Wang, L.; Peng, Z. Solving energy-efficient distributed job shop scheduling via multi-objective evolutionary algorithm with decomposition. *Swarm Evol. Comput.* **2020**, *58*, 100745. [CrossRef]
24. Şahman, M.A. A discrete spotted hyena optimizer for solving distributed job shop scheduling problems. *Appl. Soft. Comput.* **2021**, *106*, 107349. [CrossRef]
25. Lu, P.; Wu, M.C.; Tan, H.; Peng, Y.-H.; Chen, C. A genetic algorithm embedded with a concise chromosome representation for distributed and flexible job-shop scheduling problems. *J. Intell. Manuf.* **2018**, *29*, 19–34. [CrossRef]
26. Du, Y.; Li, J.; Luo, C.; Meng, L. A hybrid estimation of distribution algorithm for distributed flexible job shop scheduling with crane transportations. *Swarm Evol. Comput.* **2021**, *62*, 100861. [CrossRef]
27. Chaouch, I.; Driss, O.B.; Ghedira, K. A novel dynamic assignment rule for the distributed job shop scheduling problem using a hybrid ant-based algorithm. *Appl. Intell.* **2018**, *49*, 1903–1924. [CrossRef]
28. Taillard, E. Benchmarks for basic scheduling problems. *Eur. J. Oper. Res.* **1993**, *64*, 278–285. [CrossRef]
29. Bierwirth, C. A generalized permutation approach to job shop scheduling with genetic algorithms. *Oper. Res. Spektrum* **1995**, *17*, 87–92. [CrossRef]
30. Wang, G.; Li, X.; Gao, L.; Li, P. Energy-efficient distributed heterogeneous welding flow shop scheduling problem using a modified MOEA/D. *Swarm Evol. Comput.* **2021**, *62*, 100858. [CrossRef]
31. Storn, R. Differrential evolution a simple and efficient adaptive scheme for global optimization over continuous spaces. *Tech. Rep. Int. Comput. Sci. Inst.* **1995**, *11*, 20000924895.
32. Wang, G.; Gao, L.; Li, X.; Li, P.; Tasgetiren, M.F. Energy-efficient distributed permutation flow shop scheduling problem using a multi-objective whale swarm algorithm. *Swarm Evol. Comput.* **2020**, *57*, 100716. [CrossRef]
33. Dorigo, M.; Birattari, M.; Stutzle, T. Ant colony optimization. *IEEE Comput. Intell. Mag.* **2006**, *1*, 28–39. [CrossRef]
34. Peng, K.; Pan, Q.; Gao, L.; Li, X.; Das, S.; Zhang, B. A multi-start variable neighbourhood descent algorithm for hybrid flowshop rescheduling. *Swarm Evol. Comput.* **2019**, *45*, 92–112. [CrossRef]