Hindawi
Complexity
Volume 2020, Article ID 3450180, 15 pages
https://doi.org/10.1155/2020/3450180

Research Article

An Improved Genetic-Shuffled Frog-Leaping Algorithm for Permutation Flowshop Scheduling

Peiliang Wu,[1,2,3,4] Qingyu Yang,[1] Wenbai Chen,[5] Bingyi Mao,[1,3] and Hongnian Yu[4]

[1]*School of Information Science and Engineering, Yanshan University, Qinhuangdao 066004, China*
[2]*State Key Laboratory of Management and Control for Complex Systems, Institute of Automation, Chinese Academy of Sciences, Beijing 100190, China*
[3]*The Key Laboratory for Computer Virtual Technology and System Integration of Hebei Province, Qinhuangdao 066004, China*
[4]*School of Engineering and the Built Environment, Edinburgh Napier University, Edinburgh EH10 5DT, UK*
[5]*School of Automation, Beijing Information Science & Technology University, Beijing 100101, China*

Correspondence should be addressed to Hongnian Yu; yu61150@ieee.org

Received 27 June 2020; Revised 11 September 2020; Accepted 16 October 2020; Published 28 November 2020

Academic Editor: Zhile Yang

Due to the NP-hard nature, the permutation flowshop scheduling problem (PFSSP) is a fundamental issue for Industry 4.0, especially under higher productivity, efficiency, and self-managing systems. This paper proposes an improved genetic-shuffled frog-leaping algorithm (IGSFLA) to solve the permutation flowshop scheduling problem. In the proposed IGSFLA, the optimal initial frog (individual) in the initialized group is generated according to the heuristic optimal-insert method with fitness constrain. The crossover mechanism is applied to both the subgroup and the global group to avoid the local optimal solutions and accelerate the evolution. To evolve the frogs with the same optimal fitness more outstanding, the disturbance mechanism is applied to obtain the optimal frog of the whole group at the initialization step and the optimal frog of the subgroup at the searching step. The mathematical model of PFSSP is established with the minimum production cycle (makespan) as the objective function, the fitness of frog is given, and the IGSFLA-based PFSSP is proposed. Experimental results have been given and analyzed, showing that IGSFLA not only provides the optimal scheduling performance but also converges effectively.

1. Introduction

With the advancement of Industry 4.0, the *demographic-dividend* is gradually replaced by the *technology-dividend* [1, 2]. In the flowshop, designing an intelligent scheduling algorithm can effectively improve manufacturing systems and production efficiency of enterprises. Although scheduling is a very active field with a high practical relevance, there are still many challenging problems in the flexible manufacturing systems of Industry 4.0.

Among these challenging problems, the permutation flowshop scheduling problem (PFSSP) has been researched for more than half a century due to its complexity. It can be described as N jobs that are processed on M different machines in the same order. The processing time of the n-th job on the m-th machine is known in advance and fixed. The task is to solve the processing order of each job so that the objective function (generally refers to the time when the last job is processed on the last machine) is optimal [3–6]. Nowadays, there are a variety of dynamic factors in a highly intelligent flowshop. A reasonable scheduling method can control the production process, so that enterprises should effectively face the unexpected situation in production and processing to maximize the benefits of enterprises [4–6]. The most commonly used scheduling index is the minimum production cycle, also called as makespan, referring to the minimum time to complete the processing of all jobs [7–11].

When the number of jobs is small, the PFSSP can be solved by the deterministic solution methods [3], such as dynamic programming method and branch and bound method [12]. PFSSP becomes an NP-hard problem when the number of jobs is large [13]. It is a challenging task to solve

mainly because that a machine needs to process multiple jobs, and a job needs to be sequentially processed on multiple machines [14, 15]. The increase in the number of jobs and machines will make the optimal solution process very complicated, and the solution will be more difficult. The metaheuristic algorithm [16, 17] provides a feasible solution to the NP problem which is difficult to be solved by traditional optimization algorithm, and this kind of algorithms has been widely used in PFSSP [18–20]. The study in [21] uses genetic algorithm to solve PFSSP. The authors in [22–24] study the simulated annealing algorithm. The authors in [25, 26] study the particle swarm optimization. The authors in [27–29] study the artificial bee colony algorithm. However, each single metaheuristic algorithm has its disadvantage and some combination approaches have been proposed.

The genetic-shuffled frog-leaping algorithm (GSFLA) is a swarm intelligence optimization algorithm that simulates the process of frog foraging behavior [30–34]. It combines the advantages of the memetic algorithm (MA) based on memetic evolution [35] and the particle swarm optimization algorithm (PSO) based on swarm behavior [36]. GSFLA can realize the sharing and exchanging of group information and has the characteristics of scattered search and global information exchange. Now, GSFLA has been applied to solve generator maintenance scheduling (2015), time-optimal traveling salesman problem (2018), text document clustering (2019), and neural network structure optimization (2020).

In this paper, we propose an improved GSFLA called as IGSFLA and apply it to solve the PFSSP. This paper has three contributions: (1) a heuristic optimal-insert method with fitness constrain is applied to generate the frog (individual) group; (2) the crossover mechanism is applied to both the subgroup and the global group to avoid the local optimal solutions and make the evolution faster; (3) the disturbance mechanism is applied both in the initialization step and the local searching step to evolve the frogs with the same optimal fitness.

The remainder of the paper is structured as follows: Section 2 gives a brief overview of related work that deals with the PFSSP. The details of the IGSFLA algorithm and IGSFLA-based PFSSP are presented in Section 3. The experimental results and analysis are reported in Section 4 and in Section 5 by the conclusion.

2. Related Work

The manufacturing problems can be classified according to different characteristics, for example, the number of machines (one machine and parallel machines), the job characteristics (preemption allowed or not and equal processing times), and the number of objective (single objective usually the makespan, biobjective, and multiobjective). When each job has a fixed number of operations requiring different machines and all jobs share the same route, we are dealing with a flowshop scheduling problem. If each machine has to process the jobs in the same order, the problem is named as permutation flowshop scheduling problem (PFSSP).

Up to now, lots of methods to provide exact or approximate solutions have been presented for the PFSSP over the last 60 years, including reinforcement learning based method [37]. Ruiz has presented a review of approximate methods for PFSSP, including almost heuristics and metaheuristics with the makespan criterion [38]. Among these solution approaches, metaheuristic approaches have become more popular, but according to [38], the current state-of-the-art approach is far from easy to identify. In this section, we mainly list and analysis some of them related to our work.

2.1. Genetic Algorithms and Utilization to PFSSP. Genetic algorithm (GA) is originated from the computer simulation study of the biological system [21, 39]. It imitates the mechanism of biological evolution in nature, borrows from Darwin's theory of evolution and Mendel's theory of heredity, and is essentially an efficient, parallel, and global search method, which can automatically acquire and accumulate knowledge about the search space during the search process, and adaptively control the search process to obtain the optimal solution. GA mainly consists of three operations, namely, selection, crossover, and mutation, which are performed by calculating the fitness of the group to evolve and produce new frogs continuously.

Zhang et al. [40] proposed HGA-RMA for the PFSSP and achieved optimal performance at that time. However, due to the slow convergence speed of GA, it is still easy to fall into local optimal solutions. This issue inspires researchers to combine the genetic algorithm with other algorithms to solve PFSSP. The characteristics of noncompact flowshop scheduling plans in manufacturing enterprises are analyzed, and a scheduling strategy based on the nondominated sorting genetic algorithm (NSGA) is proposed in [21]. NSGA can guarantee the diversity and evolutionary effect of the group in the multiobjective scheduling model of noncompact flowshop.

A hybrid algorithm combining genetic algorithms and generative adversarial networks (GAN) is proposed to solve scheduling problems in [30]. The algorithm uses GAN to minimize sample information. Compared with traditional optimization algorithms, the algorithm can avoid premature local optimal solutions.

Genetic simulated annealing algorithm is proposed in [41] and improves flowshop scheduling with a makespan criterion, but the improvements are not satisfied.

Recently, Kurdi [42] combines the genetic algorithm, simulated annealing, and NEH to construct a memetic algorithm with novel semiconstructive evolution operators to solve the permutation flowshop scheduling problem, and the proposed MASC can be considered as one of the best-so-far methods for PFSSP.

2.2. Shuffled Frog-Leaping Algorithm. The idea of the shuffled frog-leaping algorithm (SFLA) is inspired from a group of frogs with differences among frogs living in a certain area. Each frog with the same structure represents a solution, and the frog group constitutes the solution space. The whole frog group is divided into several subgroups, and

each of which has its own scheduling strategies and executes local search strategies in a certain way. Each frog in the subgroup has its own scheduling strategies, which affect other frogs in and beyond this subgroup. The frogs can exchange information in a global scope.

SFLA combines particle swarm optimization (PSO) [43] based social behavior and shuffled complex evolution (SCE) [44] based complex search. As a kind of deterministic strategies, PSO allow the algorithm to effectively use response information to guide the heuristic search [42]. Meanwhile, as a kind of stochastic strategies, SCE ensures the flexibility and robustness of the search mode.

Independently, SCE and PSO have been applied to solve the PFSSP, called as SCEOL [45] and PSOENT [46] correspondingly. When the number of jobs and machines are less than 20, PSOENT obtains a better performance than GA-based HGA-RMA [40] and vice versa.

SFLA has been applied to multiobjective flexible job shop scheduling problem (MOFJSSP), and one of an approximate optimal solution can be obtained by iterating the global search several times [47]. Another solution can be obtained with an improved SFLA under four types of energy consumption [48].

Besides, the memeplex grouping SFLA is proposed and applied to solve the distributed two-stage hybrid flowshop scheduling problem (DTHFSSP) in a multifactory environment to minimize manufacturing time and the amount of delayed work [32].

Above all, SFLA has been applied to deal with MOFJSSP and DTHFSSP. Its performance to deal with PFSSP will be given in our experiments.

2.3. Genetic-Shuffled Frog-Leaping Algorithm. Because of the deficiency of SFLA itself, researchers have proposed some approaches that combine SFLA with other heuristic methods to solve different problems. Among them, the genetic-shuffled frog-leaping algorithm (GSFLA) has been proposed by combining GA and SFLA. In the local search of GSFLA, the concept of crossing between the optimal frog and the worst one is adopted to replace the *jump* operation in the traditional SFLA, to complete the evolution of the worst frog. At the same time, to avoid premature convergence of the algorithm, the idea of frog mutation is introduced to increase the diversity of the group. After the mutation, a comparison with the original frog is performed to avoid losing the optimal frog. In summary, the local search consists of the crossover of different frogs and the selective mutation of frog frogs.

As mentioned above, GSFLA has been applied to solve scheduling and clustering problems. G. Giftson Samuel proposed a hybrid PSO-based GA and hybrid PSO-based SFLA for solving long-term generation maintenance scheduling problem by considering a security constrained model [49]. Zhang et al. [50] proposed a hybrid SFLA-GA to solve the time-optimal traveling salesman problem (TOTSP) to reflect the change of traffic over time. The hybrid SFLA-GA has shown stronger search capability and fast convergence speed.

Alhenak and Hosny [51] propose a genetic frog-leaping algorithm for text document clustering in order to extract useful information from large collections of documents. In their work, the GA performs feature selection and the SFLA performs clustering. While the proposed algorithm should require longer computational time.

Inspired by above work, we try to solve the PFSSP with GSFLA. While as shown in our experiments, the traditional GSFLA is not good enough to solve the PFSSP, and in this paper, we propose an improved GSFLA and apply it to solve the PFSSP better.

3. Improved GSFLA for PFSSP

3.1. Improved GSFLA. We improve the traditional GSFLA by improving the initialization step and optimizing the crossover and disturbance mechanisms. Our crossover mechanism utilizes three traditional crossover operators. However, comparing with the traditional GSFLA crossover mechanism between the optimal frog and the worst one, we crossover all frogs in the subgroup with the optimal frog of this subgroup. Moreover, the frog of the subgroup will also crossover with the global optimal frogs to avoid local optimal solutions. Our mutation mechanism utilizes three traditional mutation operators. Comparing with the traditional GSFLA, we introduce frog disturbances to evolve the optimal frogs that have the same fitness in the subgroup. The essence of disturbances is mutations, and frogs with disturbances can explore better results.

The IGSFLA includes heuristic initialization, subgroup division, search, and shuffle.

3.1.1. Heuristic Initialization. Initialization is the first step of both traditional GSFLA and our IGSFLA to generate the frogs as a group and then to divide it into different subgroups. Here, each frog (also called as individual) presents a candidate scheduling solution of PFSSP.

For the traditional initialization of GSFLA and SFLA, frogs are initialized in a random way. Considering that the number of frogs is far less than the size of solution space, the initialized frog group is hard to simulate the optimal solution.

Considering that if the initialized frogs contain one frog that is close to the optimal solution, then the other group individuals will evolve toward to the optimal solution at high probability. In order to make the optimal initialized frogs close to the optimal solution, in this paper, we present the following optimal-insert-based heuristic initialization with optimal-fitness constraint to generate the optimal initialized frog of the whole group.

Generate the optimal initialized frog as follows:

Step 1. Choose the first item of the jobs as the first gene to form the gene queue of a frog.

For each following item in the jobs,

Step 2. Insert one job as a new gene into different positions of the gene queue to form different updated gene queues of the frog.

Step 3. Calculate each fitness of the updated gene queues.

Step 4. Sort the gene queues and select the one with optimal fitness.

Step 5. If there are several gene queues with the same optimal fitness (as called multioptimal frogs), then utilize disturbance operators to evolve them and go to *Step 3*.

End for

Step 6. Select the gene queue with optimal fitness as the optimal initialized frog.

Generate the other initialized frogs as follows:

A random method is utilized to generate the other initialized frogs of the group.

3.1.2. Division of Subgroups. The traditional division of subgroups is cutting the whole initialized frog group simply according to the size of subgroup. In this paper, to keep the variety of frogs, distribute different level frogs into each subgroup with the following division strategies:

Step 1. Calculate the fitness of the initialized frogs in the whole group.

Step 2. Sort the frogs according to the fitness in descending order.

Step 3. Record the frog with optimal fitness as P_x which denotes the current optimal solution of PFSSP.

Step 4. Distribute the sorted frogs into the each subgroup according to a snake-order. If the number of subgroups is *popnum_sub*, for the i-th frog in the group, calculate MOD$(i, popnum_sub)$, where MOD$(\cdot)$ is the modulus operation. When MOD$(i, popnum_sub)$ is not zero, the i-th frogs will be distributed into the subgroup with number MOD$(i, popnum_sub)$, otherwise, into the subgroup with number $popnum\_sub$.

3.1.3. Search Process of IGSFLA. The search process includes local search and global search.

Local Search in Each Subgroup.

Step 1. Sort the frogs in the subgroup according to their fitness in descending order.

Step 2. Crossover:

(1) Selection: select frog P_b with optimal fitness in the subgroup, and represent other frogs as P_i.

(2) Crossover: other frogs perform one of the crossover operations mentioned in following Section 3.2 randomly with P_b to produce offspring P_{F1}. P_b performs one of the crossover operations randomly on P_x to produce offspring P_{F2} to ensure that each round of local search can learn from the global optimal frog.

(3) Judgment:

If $f_{PF1} > f_{Pi}$, P_{F1} replaces P_i
else, P_i and P_x crossover to produce P_{F3}
If $f_{PF3} > f_{Pi}$, P_{F3} replaces P_i
else, stochastic feasible solution replaces P_i
If $f_{PF2} > f_{Pb}$, P_{F2} replaces P_b
else, P_b remains unchanged

Step 3. Mutation:

(1) Selection: calculate the probability of mutation for each frog. Set the range of probability γ as $\gamma \in [\gamma_{\min}, \gamma_{\max}]$, the maximum fitness of the subgroup is recorded as $f_{\max}$, the average fitness is recorded as f_{avg}, and calculate γ according to the following formula:

$$\gamma = \begin{cases} \gamma_{\max} - (\gamma_{\max} - \gamma_{\min}) \times \dfrac{(f - f_{\text{avg}})}{(f_{\max} - f_{\text{avg}})}, & f > f_{\text{avg}}, \\ \gamma_{\max}, & f \le f_{\text{avg}}. \end{cases} \tag{1}$$

(2) Mutation: perform one of the mutation operations (mentioned in Section 3.3) randomly to produce offspring.

(3) Judgment:

If $f_{P_{\text{mut}}} > f_{P_{\text{ori}}}$, P_{mut} replaces P_{ori}
else, P_{ori} remains unchanged

where P_{ori} denotes the original frog and P_{mut} denotes the mutated frog.

Step 4. Frog disturbance to the multioptimal frog:

(1) Selection: select each frog P_b in the multioptimal frogs in each subgroup.

(2) Mutation: perform one of the mutation operations randomly to produce offspring.

(3) Judgment:

If $f_{P_{\text{dis}}} > f_{P_b}$, P_{dis} replaces P_b
else, P_b remains unchanged

where P_{dis} denotes the disturbed frog

Step 5. Reorder subgroups and iterates:

Global Search.

Step 1. Perform local search on subgroups (as shown above), and this step mainly applied three traditional operations: crossover, mutation, and disturbance.

Step 2. Shuffle the subgroups and recalculate P_x.

Step 3. Verify whether the convergence condition is satisfied. If so, output the result; otherwise, return to *Step 3*.

3.2. Detailed Operation of Crossover. Three traditional crossover operators are applied in this paper: position-based

crossover, sequence-based crossover, and loop-based crossover.

3.2.1. Position-Based Crossover

(1) Select the 1/2 genes in parent frog P_{p1} randomly

(2) Record and sort the positions of unselected genes in P_{p1}

(3) In parent frogs P_{p2}, find the corresponding genes which are the same as the selected genes in parent frogs P_{p1}

(4) Record and sort the positions of unselected genes in P_{p2}

(5) Crossover the unselected genes according to the sorted positions in (2) and (4)

To demonstrate the position-based crossover operating, an example of schematic diagram is given in Figure 1. As shown in the figure, in order to generate offspring P_{o1} and P_{o2} from the parent frogs P_{p1} and P_{p2}, firstly, the genes of [g2, g4, g5] in P_{p1} are selected randomly with their positions [2, 4, 5], and so the unselected genes in P_{p1} are [g1, g3, g6] with their positions [1, 3, 6]. According to the selected [g2, g4, g5] in P_{p1}, we find the same [g2, g4, g5] with their positions [2, 3, 5] in P_{p2}, and then the unselected positions are [1, 4, 6] with genes [g3, g6, g1] in P_{p2}; we crossover the corresponding genes in unselected [g1, g3, g6] of P_{p1} and unselected [g3, g6, g1] of P_{p2}, and then we obtain the offspring P_{o1} and P_{o2}.

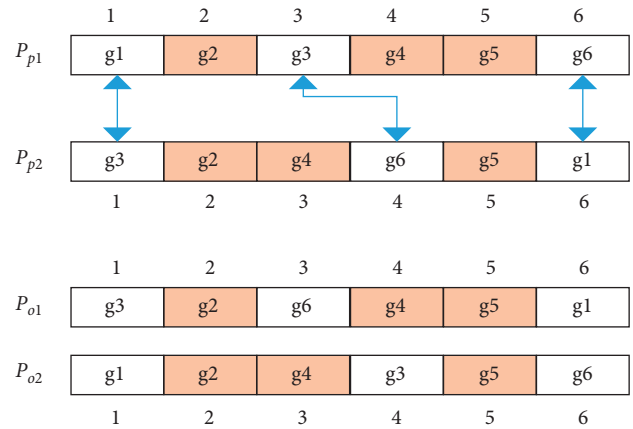

Figure 1: Schematic diagram of position-based crossover.

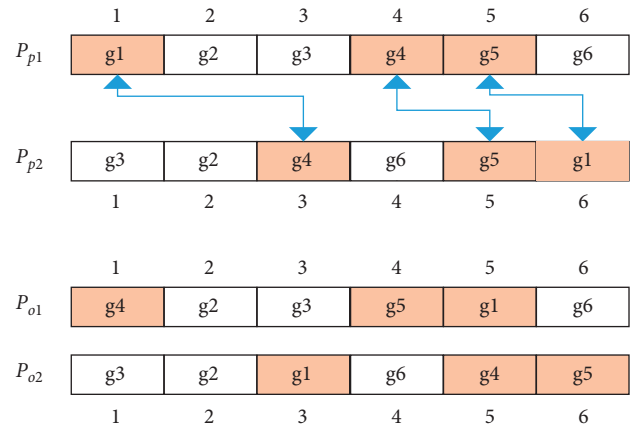

Figure 2: Schematic diagram of sequence-based crossover.

3.2.2. Sequence-Based Crossover

(1) Select the 1/2 genes in parent frog P_{p1} randomly

(2) Record and sort the positions of selected genes in P_{p1}

(3) In parent frogs P_{p2}, find the corresponding genes which are the same as the selected genes in parent frogs P_{p1}

(4) Record and sort the positions of selected genes in P_{p2}

(5) Crossover the selected genes according to the sorted positions in (2) and (4)

Schematic diagram of the sequence-based crossover is shown in Figure 2.

3.2.3. Loop-Based Crossover

(1) Initialize order number j as 1 and position queue K as NULL.

(2) While the gene at position j in P_{p2} differs from the first gene in P_{p1},

update j with the position order of gene in P_{p1} which is the same to the one at position j in P_{p2}

put j into position queue K

End while

(3) Generate P_{o1} by gathering genes in the position queue K in P_{p1} and genes beyond the position queue K in P_{p2}. Generate P_{o2} by gathering genes in the position queue K in P_{p2} and genes beyond the position queue K in P_{p1}.

Schematic diagram of loop-based crossover is shown in Figure 3.

3.3. Detailed Operation of Mutation. Three traditional mutation operators are applied in this paper: inverted mutation, transformation mutation, and insertion mutation.

3.3.1. Inverted Mutation

(1) Randomly select two genes on the frog

(2) Reverse the order of the genes between the two selected genes

Schematic diagram of inverted mutation is shown in Figure 4.

3.3.2. Transformation Mutation

(1) Randomly select two genes on the frog

(2) Swap positions for the two selected genes

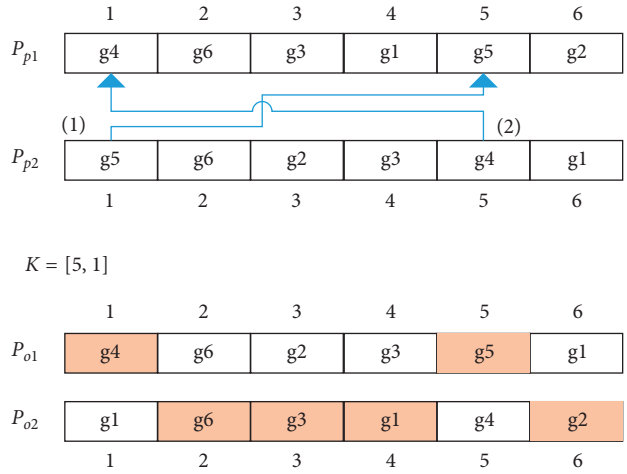

Figure 3: Schematic diagram of loop-based crossover.

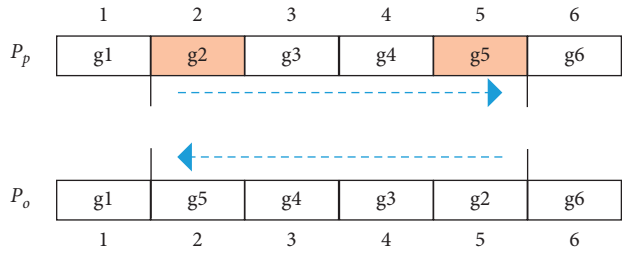

Figure 4: Schematic diagram of inverted mutation.

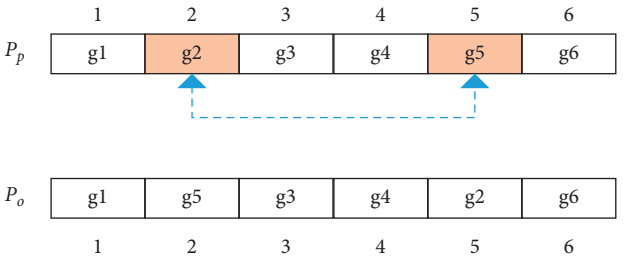

Figure 5: Schematic diagram of transformation mutation.

Schematic diagram of transformation mutation is shown in Figure 5.

3.3.3. Insertion Mutation

(1) Randomly select two genes on the frog

(2) Forward insert: insert the second selected gene in front of the first gene

(3) Backward insert: insert the first selected gene after the second gene

Schematic diagrams of forward insert and backward insert are shown in Figures 6 and 7.

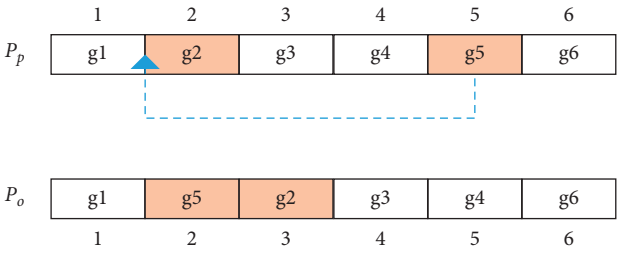

Figure 6: Schematic diagram of forward insert.

3.4. The Mathematical Model of PFSSP. PFSSP refers to the processing flow of N jobs $\{\text{Job}_n\}_{n=1}^{N}$ on M machines $\{\text{Machine}_m\}_{m=1}^{M}$. The processing time of each job on each machine is known, so the scheduling algorithm needs to give the processing order of each job on each machine to fit the following additional hypotheses.

Hypotheses about jobs are as follows:

The processing order of each job on each machine is the same and known
Each job can only be processed on one machine at a time
Each job is independent of each other

Hypotheses about machines are as follows:

Each machine can only process one job at a time
Each machine is independent of each other

Hypotheses about production process are as follows:

Each job has no processing priority
The transmission time of a job between machines is considered negligible

Mathematical model:

Let the PFSSP processing order be represented as a permutation $\lambda = \{\lambda(1), \lambda(2), \ldots, \lambda(N)\}$, where $\lambda(n)$ denotes the index of job arranged at the n-th position of λ. According the following set of recursive equations, we could calculate the completion time of each job $t_{\lambda(n),m}, n = 1, 2, \ldots, N, m = 1, 2, \ldots, M$ [42]:

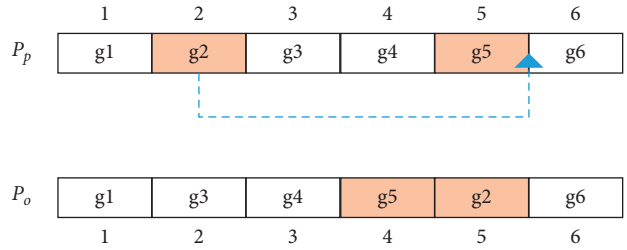

Figure 7: Schematic diagram of backward insert.

$$\begin{aligned}
t_{\lambda(1),1} &= s_{\lambda(1),1},\\
t_{\lambda(1),m} &= t_{\lambda(1),m-1} + s_{\lambda(1),m}, \quad m = 2,3,\ldots,M,\\
t_{\lambda(n),1} &= t_{\lambda(n-1),1} + s_{\lambda(n),1}, \quad n = 2,3,\ldots,N,\\
t_{\lambda(n),m} &= \max\left\{t_{\lambda(n-1),m}, t_{\lambda(n),m-1}\right\} + s_{\lambda(n),m}, \quad n = 2,3,4,\ldots,N; m = 2,3,4,\ldots,M,
\end{aligned} \tag{2}$$

$$F\left(\lambda^*\right) \le F(\lambda), \tag{3}$$

$$F(P) = -F(\lambda) = -t_{\lambda(N),M}. \tag{4}$$

where $t_{\lambda(n),k}m$ denotes the time when the n-th job is processed on the m-th machine. $s_{\lambda(n),m}$ denotes the processing time of the n-th job on the m-th machine. The makespan $F(\lambda) = t_{\lambda(N),M}$ denotes the time when the last job is processed on the last machine.

Objective function:

where $F(\lambda) = t_{\lambda(N),M}$.

The definition of fitness:

As mentioned above, the fitness function is needed to determine how good a frog is. For the frog P in PFSSP, the fitness function is defined simply as the minus of makespan:

3.5. IGSFLA-Based PFSSP. The IGSFLA-based PFSSP includes group initialization, group division, local search, and global search. Local search is mainly to complete the evolution of the frogs in the subgroup. Global search is mainly to search for the most adaptable frogs in the group and to complete the group division and merge the subgroups. In this paper, the process is described as follows:

Step 1. Mathematical modeling and parameter setting: construct the time matrix $T = (M, N)$ according to the PFSSP. To ensure that the time matrix is the same when the number of machines and jobs are the same, the *state* parameter is introduced. Initialize the number of subgroups *popsize_num* and the number of frogs in a subgroup *popsub_num*, so the size of the whole group is $popsize = popsize\_num \times popsub\_num$.

Step 2. Heuristic group initialization: initialize the frog group according to Section 3.1.1. Sort the fitness of frogs according to the fitness, and represent the globally optimal frog as P_x.

Step 3. Group division: divide the frog into several subgroups utilizing the snake-order-grouping method according to Section 3.1.2.

Step 4. Local search according to Section 3.1.3.

Step 5. Global search: shuffle all subgroups and sort all frogs in order of fitness from largest to smallest, and record the global optimal frog P_x.

Step 6. Go back to *Step 3* until the conditional output is reached.

4. Experiments and Analysis

In this section, firstly, we test the parameters that need to be set for the proposed IGSFLA, which are the number of subgroup iterations c and the number of frogs in the subgroup *popsize_num*. Then, we compare the results of random initialization and heuristic initialization with optimal-insert in GSFLA and IGSFLA to demonstrate the effectiveness of the proposed heuristic initialization with the optimal-insert method. Thirdly, PFSSP based on IGSFLA is compared with IGSFLA-no-disturbance (which lacks the disturbance mechanism compared to IGSFLA), GA, SFLA, and GSFLA, to show the effectiveness and convergence of disturbance mechanism. Finally, we compare our IGSFLA with two kinds of state-of-the-art methods on open datasets.

The experiments are executed on a laptop with Windows 10 OS, Intel Core i5-3230M CPU and 8G RAM.

4.1. Algorithm Parameter Testing

4.1.1. The Number of Subgroup Iterations. To test the number of subgroup iterations c, the number of subgroups is fixed as $popsize\_num = 30$ and the number of frogs in a subgroup as $popsub\_num = 30$. 9 kinds of combinations with N jobs and M machines are set up as shown in Table 1. The experiment is repeated 10 times under 5 conditions of $c = 15$, 30, 50, 75, and 100. Keeping the optimal value of each experiment, the averages of each 10 experiments are calculated and shown in Table 1. Because the NP-hard nature, we do not need to find the real optimal makespan. For the algorithms in our experiments, the makespan obtained after iterating 50 times is considered as the optimal value.

Through analyzing the experimental results in Table 1, it can be found that the value of c has an impact on the algorithm performance. If c is set too small, too many global exchanges will result in poor accuracy of the algorithm. However, if c is too large, the global mixing times will be reduced correspondingly, the algorithm will lose the advantage of global search, the subgroup frogs will not be able to communicate well with the global optimal solution, and the increase of local search times will lead to the increase of computation, making the algorithm less efficient. In the above parameter combination, when the number of machines and the number of jobs are small, the value c is not so important. However, when the number of machines and the number of jobs become large, a large c will cause a lot of unnecessary calculations; otherwise, a small c will reduce the calculations more effectively. The optimal c depends on the number of machines and jobs. With the increasing in the number of jobs and machines, the optimal c should be increased. In our experiments, the proposed algorithm can achieve optimal when $c = 50$ for 9 kinds of combinations in Table 1.

4.1.2. The Number of Frogs in the Subgroup. To test the number of frogs in the subgroup $popsub\_num$, the total group size is fixed as $popsize = 900$ and the number of subgroup iterations as $c = 50$. 5 kinds of job and machine combinations are set up, and the experiment is repeated 10 times under conditions of $popsub\_num = 10$, 20, 30, 40, 50, and 60. Keeping the optimal value of each experiment, the averages of the results of 10 experiments are calculated and shown in Table 2.

Through analyzing the experimental results in Table 2, it can be found that the proposed IGSFLA could perform better when $popsub\_num$ is between 30 and 40. The results show that $popsub\_num$ is not as large as possible. If $popsub\_num$ is too small, the subgroup cannot adequately communicate with each other, and the algorithm loses the advantage of local search. If $popsub\_num$ is too large, the algorithm is easy to fall into local optimization.

4.1.3. The Heuristic Initialization with Optimal-Insert Method. To test the effect of heuristic initialization with the optimal-insert method, the total group size is fixed as $popsize = 900$, the number of subgroup iterations as $c = 50$, and the number of frogs in a subgroup as $popsub\_num = 30$. 5 kinds of jobs and machine combinations are set up and the experiment is repeated 10 times under the traditional random initialization and our heuristic initialization. Keeping the optimal value of each experiment, the averages of the results of 10 experiments are calculated and shown in Table 3.

From Table 3, it can be found easily that the heuristic initialization with the optimal-insert method of fitness constraint could generate high-quality frogs, especially the initialized frog that has the optimal fitness. Based on the high-quality initialization, it is much easier to obtain better makespan in both traditional GSFLA and our IGSFLA.

4.2. Optimization Result Testing.

First, the number of subgroups is set as $popsize\_num = 30$, and the number of frogs in a subgroup is set as $popsub\_num = 30$. Then, 9 kinds of combinations with N jobs and M machines are set up as shown in Table 2. The experiment is repeated 10 times using 5 different algorithms, which are GA, SFLA, GSFLA, IGSFLA-no-disturbance, and IGSFLA. Following the contrast principle, the same parameters are set in the 4 algorithms of SFLA, GSFLA, IGSFLA-no-disturbance, and IGSFLA. Keeping the optimal value of each experiment, the averages of the results of 10 experiments are calculated and shown in Table 4.

From Table 4, it can be seen that with the increase in the number of jobs and the number of machines, the proposed IGSFLA gradually shows its advantages. Comparing with GA, SFLA, and GSFLA, IGSFLA can get a better solution because of its powerful search ability. Besides, the disturbance could improve the performance obviously. As mentioned above in Sections 3.1.1 and 3.1.3, the disturbance works effectively in two places, which are the initialization step and the local searching step. This could evolve the frogs with the same optimal fitness more outstanding, so the group could simulate the optimal situation more effectively.

4.3. Algorithm Convergence.

In the actual flow workshop, the improvement of production efficiency is an overall problem. The scheduling system needs the scheduling algorithm to give an excellent scheduling plan in a short time. Therefore, the convergence of the algorithm is also very significant.

To test the convergence of IGSFLA, the number of jobs is fixed as $N = 50$, and the number of machines as $M = 20$. SFLA, GSFLA, IGSFLA-no-disturbance, and IGSFLA are set with the same parameters: $c = 50$, $popsize\_num = 30$, and $popsub\_num = 30$. Each algorithm is iterated for 50 times. While for GA, *iterated* here refers to the completion of an overall evolution, and for SFLA, GSFLA, IGSFLA-no-disturbance, and IGSFLA, *iterated* here refers to the completion of a global search. The Gantt charts of the solution with above algorithms are provided in Figures 8–12. The

Table 1: The average value of makespan under different values of c (unit: second).

$N \times M$	$c = 15$	$c = 30$	$c = 50$	$c = 75$	$c = 100$
10×5	712.51	712.51	712.51	712.51	712.51
10×10	1092.53	1092.01	1092.01	1092.56	1092.01
10×20	1659.27	1659.27	1659.27	1659.27	1659.27
20×5	1326.53	1326.53	1326.53	1326.53	1326.53
20×10	1537.18	1536.02	1499.27	1499.27	1499.27
20×20	2378.05	2372.14	2358.26	2358.26	2356.26
50×5	2832.57	2831.28	2831.53	2832.57	2831.53
50×10	3223.09	3222.34	3133.23	3133.23	3133.23
50×20	4105.62	4093.42	3903.56	3963.42	3890.75

Table 2: The average value of makespan under different values of *popsub_num* (unit: second).

$N \times M$	10	20	30	40	50	60
10×20	1659.27	1659.27	1659.27	1659.27	1659.27	1659.27
20×10	1529.86	1527.25	1499.27	1536.02	1537.18	1564.53
20×20	2370.41	2371.81	2358.26	2372.14	2378.05	2335.64
50×10	3193.02	3182.44	3133.23	3129.48	3136.75	3133.64
50×20	3928.76	3948.46	3903.56	4038.94	4002.01	3973.42

Table 3: The average value of makespan under two different initialization methods (unit: second).

$N \times M$	Random initialization + GSFLA	Heuristic initialization + GSFLA	Random initialization + IGSFLA	Heuristic initialization + IGSFLA
10×5	712.57	712.62	712.56	712.56
10×10	1094.97	1092.29	1093.56	1092.29
10×20	1659.27	1659.27	1662.63	1662.29
20×5	899.69	898.19	898.19	898.19
20×10	1521.29	1525.39	1520.10	1522.81
20×20	2363.80	2374.36	2364.89	2357.39
50×5	2830.59	2831.18	2830.95	2830.59
50×10	3146.25	3109.21	3121.39	3099.38
50×20	3954.47	3929.23	3894.31	3884.92

Table 4: The average value of makespan under different algorithms (unit: second).

$N \times M$	GA	SFLA	GSFLA	IGSFLA-no-disturbance	IGSFLA
10×5	712.59	712.64	712.57	712.56	712.56
10×10	1092.34	1098.23	1094.97	1093.83	1092.29
10×20	1662.31	1670.64	1659.27	1659.64	1662.29
20×5	899.64	904.18	899.69	904.18	898.19
20×10	1567.90	1543.32	1521.29	1527.45	1522.81
20×20	2410.29	2385.02	2363.80	2362.87	2357.39
50×5	2848.84	2862.13	2830.59	2837.47	2830.59
50×10	3332.76	3255.98	3146.25	3169.22	3099.38
50×20	4084.05	4023.12	3954.47	3965.84	3884.92

convergence graph is drawn according to the data. The results are shown in Figure 13.

For actual production tasks, the scheduling system often does not need an optimal result, but a suboptimal that can be obtained in a short time is good enough. To quantify the convergence of the algorithms, the statistical results are shown in Table 5. Here, the optimal result of GA with 86 iterations is considered as the benchmark and the number of iterations required by the other algorithms to achieve the benchmark is given. It can be seen that to achieve the same optimal result of benchmark, IGSFLA-no-disturbance needs 15 iterations, while GSFLA and IGSFLA only need 2 iterations.

Combining the convergence comparison shown in Figure 13 and Table 5, it can be seen that IGSFLA converges faster and gets better results than IGSFLA-no-disturbance, GSFLA, SFLA, and GA. Comparing with GSFLA, IGSFLA further improves the scheduling ability by optimizing the

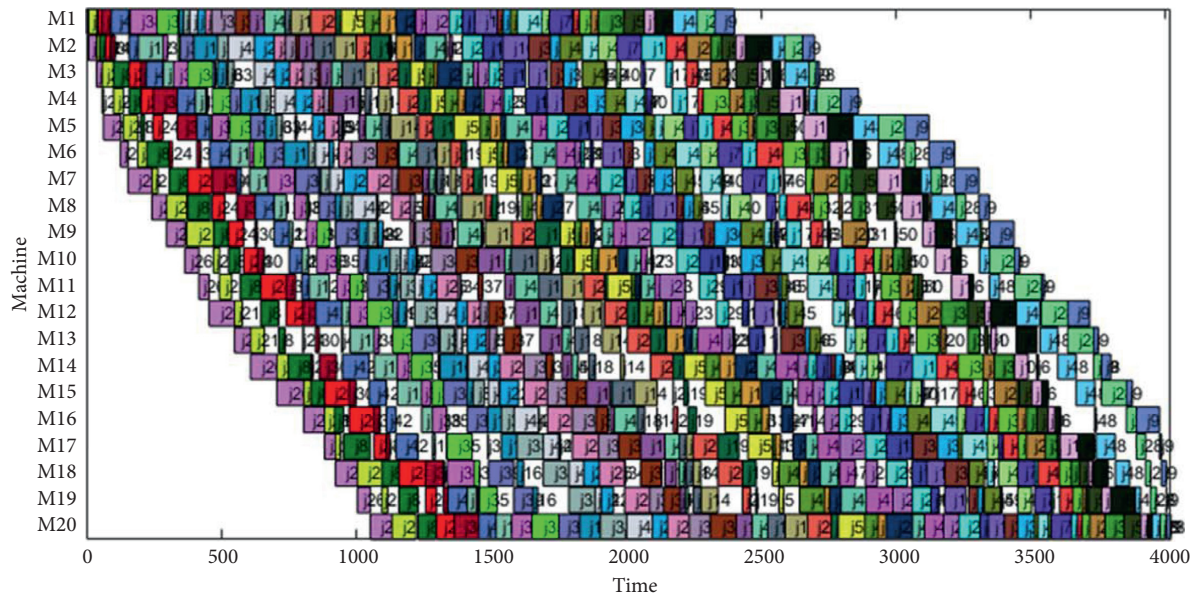

Figure 8: Gantt chart of the solution with GA (workpiece: 50, machine: 20, makespan: 4017.85).

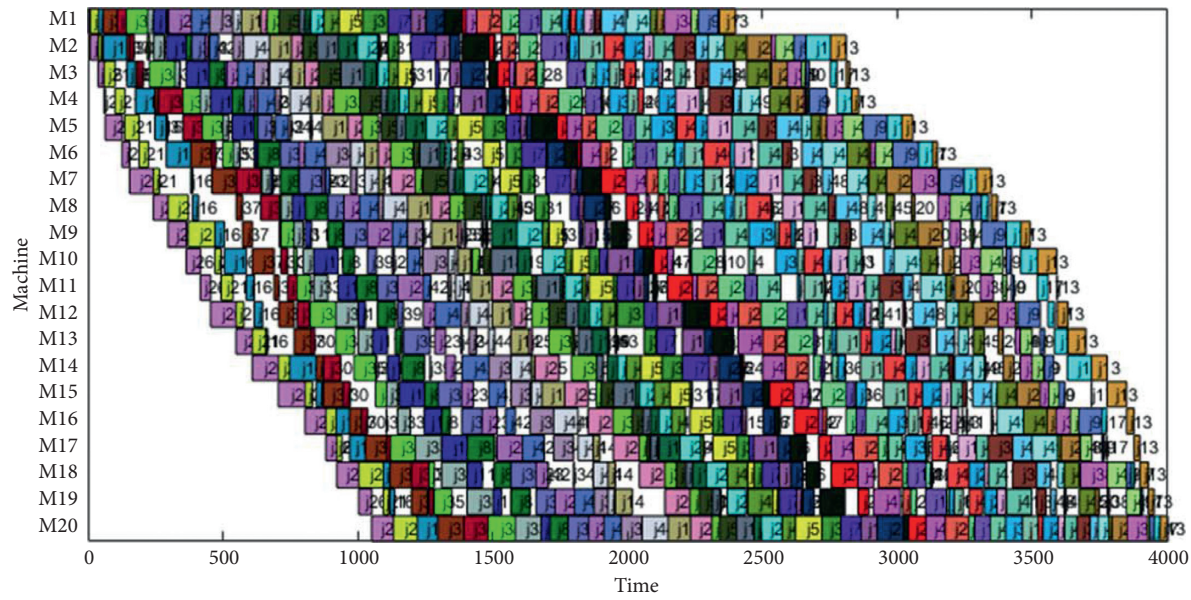

Figure 9: Gantt chart of the solution with SFLA (workpiece: 50, machine: 20, makespan: 4003.70).

initialization and improving the crossover and disturbance mechanism, converges faster, and gets better results with less iteration times.

4.4. Further Comparison and Discussion. In 1989, Taillard chose the hardest problems and their optimal solutions to construct the Taillard dataset. The new optimal solution will be updated into the dataset when it is appeared and confirmed.

Here, the scheduling results executed on the Taillard dataset are listed and compared with the proposed IGSFLA and with some other methods besides metaheuristic methods.

As shown in Table 6, our IGSFLA could perform better than the Q-learning method proposed in 2017 which belongs to a new self-learning method called as reinforcement learning. Besides, our proposed IGSFLA could nearly achieve the optimal solution of MASC in 2020. Furthermore, the IGSFLA could perform as well as MASC when the number of machines is less than 20. While with the increasing of machine number, the deficiency of IGSFLA between the state-of-the-art method appears, which should be researched in our further work.

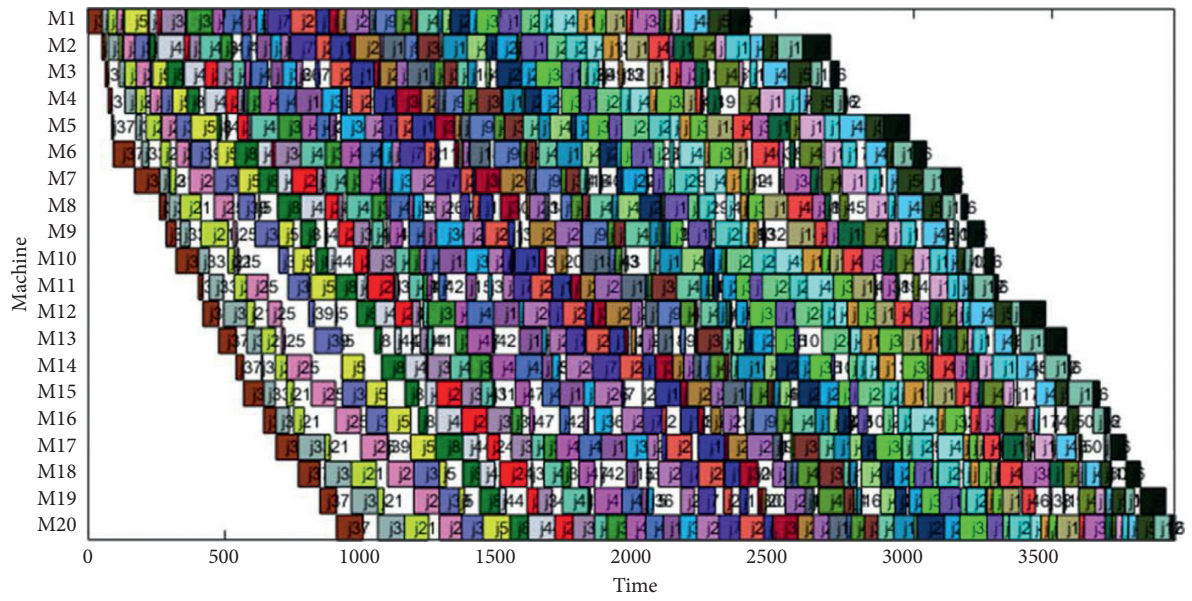

Figure 10: Gantt chart of the solution with GSFLA (workpiece: 50, machine: 20, makespan: 3948.57).

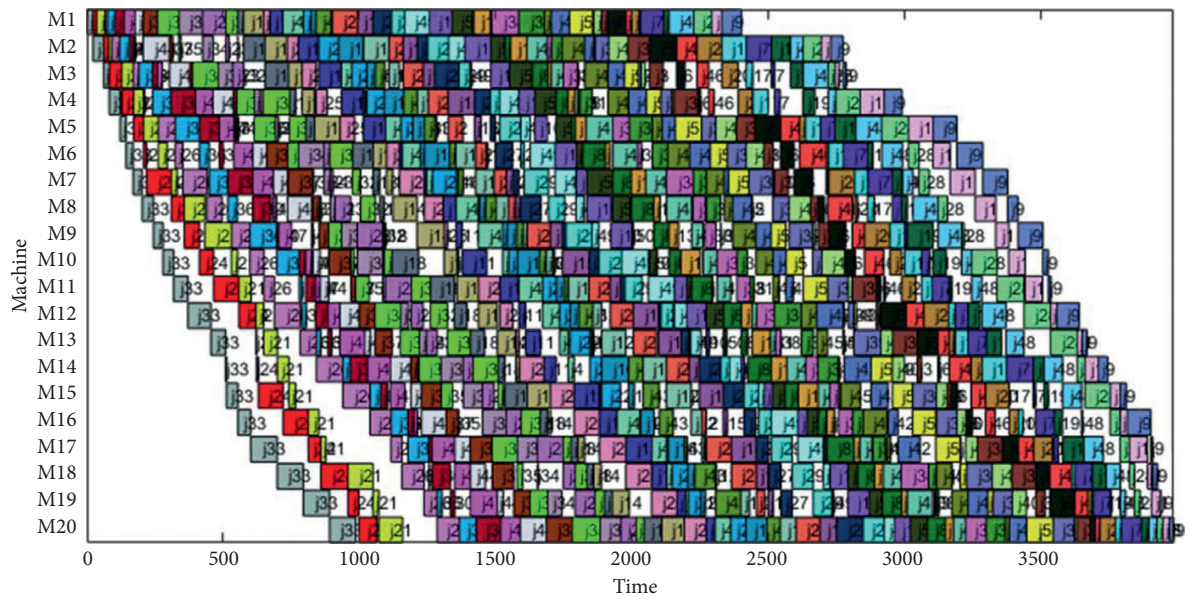

Figure 11: Gantt chart of the solution with IGSFLA-no-disturbance (workpiece: 50, machine: 20, makespan: 3986.37).

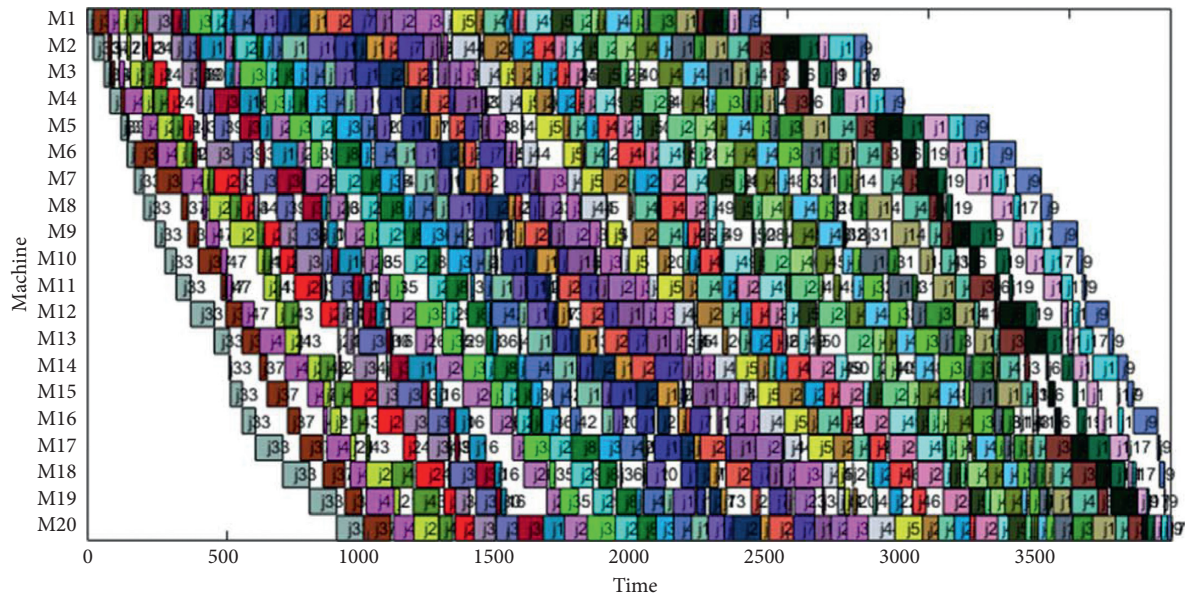

Figure 12: Gantt chart of the solution with IGSFLA (workpiece: 50, machine: 20, makespan: 3866.45).

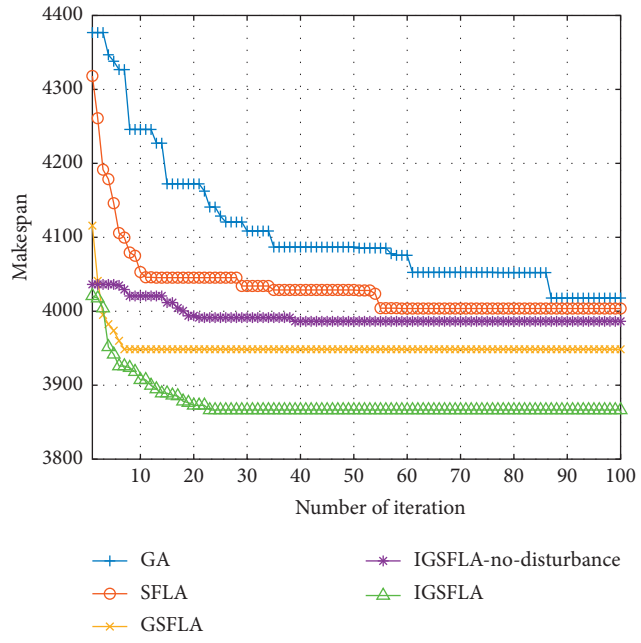

Figure 13: Convergence comparison graph (unit: second).

Table 5: Convergence comparison (unit: second).

	Optimal value	Benchmark	Iterations
GA	4017.87	4017.87	86
SFLA	4003.70	4017.87	54
GSFLA	3948.57	4017.87	2
IGSFLA-no-disturbance	3986.47	4017.87	15
IGSFLA	3866.45	4137.98	2

Table 6: Comparison between IGSFLA and some approaches on Taillard's instances (unit: second).

Dataset ($N \times M$)	Instance	Optimal value	QL (2017)	MASC (2020)	IGSFLA
20×5	Ta001	1278	1278	1278	1278
20×5	Ta005	1236	1236	1236	1236
50×5	Ta036	2829	2831	2829	2829
50×5	Ta037	2725	2728	2725	2725
20×10	Ta011	1582	1583	1582	1582
20×10	Ta017	1484	1484	1484	1484
20×20	Ta024	2223	2229	2223	2223
20×20	Ta027	2273	2295	2273	2276

5. Conclusion

Technological developments along with the emergence of Industry 4.0 call for new algorithms to solve fundamental industrial problems, especially the flowshop scheduling problem. In this paper, to improve the efficiency of permutation flowshop scheduling, the IGSFLA algorithm is proposed and applied to solve the PFSSP. In IGSFLA, to enhance the local search ability, the crossover mechanisms are optimized, and the disturbance mechanism is utilized. The mathematical model of PFSSP is established and solved based on the proposed IGSFLA algorithm. Experimental results show that, compared with other algorithms, IGSFLA could converge quickly by giving an approximately optimal value with fewer iterations and achieve the optimal scheduling solution when the number of machines is small. The following work is to improve the performance to deal with complicated PFSSP with enormous number of jobs and machines.

Data Availability

The data used to support the findings of this study are available from the corresponding author upon request.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This work was partially supported by the National Key R&D Program of China (2018YFB1308300), the European Commission Marie Skłodowska-Curie SMOOTH (smart robots for fire-fighting) project (H2020-MSCA-RISE-2016-734875), the China Postdoctoral Science Foundation (2018M631620), the Natural Science Foundation of Beijing Municipality (4202026), and the Doctoral Fund of Yanshan University (BL18007).

References

[1] E. Oztemel and S. Gursev, "Literature review of Industry 4.0 and related technologies," *Journal of Intelligent Manufacturing*, vol. 31, no. 1, pp. 127–182, 2020.

[2] H. J. Kim and J. H. Lee, "Robot task sequencing for a flexible assembly system with 3D printers," in *Proceedings of the 2017 4th International Conference on Control, Decision and Information Technologies*, pp. 1–5, Barcelona, Spain, April 2017.

[3] R. Ruiz and J. A. Vázquez-Rodríguez, "The hybrid flow shop scheduling problem," *European Journal of Operational Research*, vol. 205, no. 1, pp. 1–18, 2010.

[4] C. Lu, L. Gao, X. Li, and Q. Wang, "Energy-efficient permutation flow shop scheduling problem using a hybrid multi-objective backtracking search algorithm," *Journal of Cleaner Production*, vol. 144, pp. 228–238, 2017.

[5] M. Pan, G. Manogaran, and D. El-Shahat, "A hybrid whale optimization algorithm based on local search strategy for the permutation flow shop scheduling problem," *Future Generation Computer Systems*, vol. 85, pp. 129–145, 2018.

[6] K. Z. Mirjalili, P. N. Suganthan, Q. K. Pan, T. X. Cai, and C. S. Chong, "Discrete harmony search algorithm for flexible job shop scheduling problem with multiple objectives," *Journal of Intelligent Manufacturing*, vol. 27, no. 2, pp. 363–374, 2016.

[7] J. B. Chua, J. Xu, and J. Yang, "Bicriterion optimization for flow shop with a learning effect subject to release dates," *Complexity*, vol. 2018, p. 12, Article ID 9149510, 2018.

[8] T.-S. Yu and M. Pinedo, "Flow shops with reentry: reversibility properties and makespan optimal schedules," *European Journal of Operational Research*, vol. 282, no. 2, pp. 478–490, 2020.

[9] S. R. Hejazi and S. Saghafian, "Flowshop-scheduling problems with makespan criterion: a review," *International Journal of Production Research*, vol. 43, no. 14, pp. 2895–2929, 2005.

[10] R. Ruiz, Q.-K. Pan, and B. Naderi, "Iterated Greedy methods for the distributed permutation flowshop scheduling problem," *Omega*, vol. 83, pp. 213–222, 2019.

[11] L. Meng, C. Zhang, X. Shao, and C. Ren, "Mathematical modelling and optimisation of energy-conscious hybrid flow shop scheduling problem with unrelated parallel machines," *International Journal of Production Research*, vol. 57, no. 4, pp. 1119–1145, 2019.

[12] S. Ren, M. Liu, and C. Chu, "A branch-and-bound algorithm for two-stage no-wait hybrid flow-shop scheduling," *International Journal of Production Research*, vol. 53, no. 4, pp. 1143–1167, 2015.

[13] H. Röck, "The three-machine No-wait flow shop is NP-complete," *Journal of the ACM*, vol. 31, no. 2, pp. 336–345, 1984.

[14] Y. Yin, K. E. Stecke, and D. N. Li, "The evolution of production systems from Industry 2.0 through Industry 4.0," *International Journal of Production Research*, vol. 56, no. 1-2, pp. 848–861, 2018.

[15] A. Dolgui, D. Ivanov, and S. P. Sethi, "Scheduling in production, supply chain and Industry 4.0 systems by optimal control: fundamentals, state-of-the-art and applications," *International Journal of Production Research*, vol. 57, no. 2, pp. 411–432, 2019.

[16] Y. Sokolov, Y. P. Chen, M. X. Zhao et al., "Optimization of planning layout of urban building based on improved logit

and PSO algorithms," *Complexity*, vol. 2018, Article ID 9452813, 11 pages, 2018.

[17] Y. A. Wu, G. Z. Peng, H. W. Wang et al., "A heuristic algorithm for optimal service composition in complex manufacturing networks," *Complexity*, vol. 2019, Article ID 7819523, 20 pages, 2019.

[18] Y. Yu, H. Ma, M. Yu, and X. Chen, "Multipopulation management in evolutionary algorithms and application to complex warehouse scheduling problems," *Complexity*, vol. 2018, Article ID 4730957, 14 pages, 2018.

[19] R. Ye, "Computational complexity and solution algorithms for flowshop scheduling problems with the learning effect," *Computers & Industrial Engineering*, vol. 61, no. 1, pp. 20–31, 2011.

[20] C. W. Qu, Y. M. Fu, Z. J. Yi et al., "Solutions to no-wait flow shop scheduling problem using the flower pollination algorithm based on the hormone modulation mechanism," *Complexity*, vol. 2018, Article ID 1973604, 18 pages, 2018.

[21] W. Chen and Y. F. Hao, "Genetic algorithm-based design and simulation of manufacturing flow shop scheduling," *International Journal of Simulation Modelling*, vol. 17, no. 4, pp. 702–711, 2018.

[22] H. Guo, C. D. Li, Y. Zhang et al., "A nonlinear integer programming model for integrated location, inventory, and routing decisions in a closed-loop supply chain," *Complexity*, vol. 2018, Article ID 2726070, 17 pages, 2018.

[23] B. Naderi, R. Tavakkoli-Moghaddam, and M. Khalili, "Electromagnetism-like mechanism and simulated annealing algorithms for flowshop scheduling problems minimizing the total weighted tardiness and makespan," *Knowledge-Based Systems*, vol. 23, no. 2, pp. 77–85, 2010.

[24] E. Torabzadeh and M. Zandieh, "Cloud theory-based simulated annealing approach for scheduling in the two-stage assembly flowshop," *Advances in Engineering Software*, vol. 41, no. 10-11, pp. 1238–1243, 2010.

[25] D. Tang, M. Dai, and M. A. Salido, "Energy-efficient dynamic scheduling for a flexible flow shop using an improved particle swarm optimization," *Computers in Industry*, vol. 81, pp. 82–95, 2016.

[26] M. R. Giret, M. Singh, S. S. Mahapatra et al., "Particle swarm optimization algorithm embedded with maximum deviation theory for solving multi-objective flexible job shop scheduling problem," *International Journal of Advanced Manufacturing Technology*, vol. 85, no. 9–12, pp. 2353–2366, 2016.

[27] D. Gong, Y. Han, and J. Sun, "A novel hybrid multi-objective artificial bee colony algorithm for blocking lot-streaming flow shop scheduling problems," *Knowledge-Based Systems*, vol. 148, pp. 115–130, 2018.

[28] X. Li, Z. Peng, B. Du, W. Xu, and K. Zhuang, "Hybrid artificial bee colony algorithm with a rescheduling strategy for solving flexible job shop scheduling problems," *Computers & Industrial Engineering*, vol. 113, pp. 10–26, 2017.

[29] M. F. Guo, Q.-K. Pan, and P. N. Suganthan, "A discrete artificial bee colony algorithm for the total flowtime minimization in permutation flow shops," *Information Sciences*, vol. 181, no. 16, pp. 3459–3475, 2011.

[30] M. H. Chen, R. R. Yu, S. J. Xu et al., "An improved algorithm for solving scheduling problems by combining generative adversarial network with evolutionary algorithms," in *Proceedings of the 3rd International Conference on Computer Science and Application Engineering*, pp. 1–7, Sanya, China, October 2019.

[31] D. M. Lei and X. F. Tan, "Shuffled frog-leaping algorithm for order acceptance and scheduling in flow shop," in *Proceedings of the 35th Chinese Control Conference 2016*, pp. 9445–9450, Chengdu, China, July 2016.

[32] D. Lei and T. Wang, "Solving distributed two-stage hybrid flowshop scheduling using a shuffled frog-leaping algorithm with memeplex grouping," *Engineering Optimization*, vol. 52, no. 9, 2019.

[33] P. Kaur and S. Mehta, "Resource provisioning and work flow scheduling in clouds using augmented Shuffled Frog Leaping Algorithm," *Journal of Parallel and Distributed Computing*, vol. 101, pp. 41–50, 2017.

[34] D. Lei, Y. Zheng, and X. Guo, "A shuffled frog-leaping algorithm for flexible job shop scheduling with the consideration of energy consumption," *International Journal of Production Research*, vol. 55, no. 11, pp. 3126–3140, 2017.

[35] J. Deng and L. Wang, "A competitive memetic algorithm for multi-objective distributed permutation flow shop scheduling problem," *Swarm and Evolutionary Computation*, vol. 32, pp. 121–131, 2017.

[36] J.-q. Li, H.-y. Sang, Y.-y. Han, and K.-z. Gao, "Efficient multi-objective optimization algorithm for hybrid flow shop scheduling problems with setup energy consumptions," *Journal of Cleaner Production*, vol. 181, pp. 584–598, 2018.

[37] Y. C. Wang, Y. Martinez-Jimenez, and A. Nowe, "Q-learning algorithm performance for M-machine, N-jobs flow shop scheduling problems," *Revista Investigacion Operacional*, vol. 38, no. 3, pp. 281–290, 2017.

[38] V. Fernandez-Viagas and R. Ruiz, "A new vision of approximate methods for the permutation flowshop to minimise makespan: state-of-the-art and computational evaluation," *European Journal of Operational Research*, vol. 257, no. 3, pp. 707–721, 2017.

[39] J. Framinan, L. Li, F. Zhao, and Q. Zhao, "A multi-level optimization approach for energy-efficient flexible flow shop scheduling," *Journal of Cleaner Production*, vol. 137, pp. 1543–1552, 2016.

[40] R. Zhang, C. Maroto, and J. Alcaraz, "Two new robust genetic algorithms for the flowshop scheduling problem," *Omega*, vol. 35, no. 4, pp. 461–476, 2006.

[41] H. Wei, S. Li, H. Jiang et al., "Hybrid genetic simulated annealing algorithm for improved flow shop scheduling with makespan criterion," *Applied Science*, vol. 8, no. 12, Article ID 26212018, 2018.

[42] M. Kurdi, "A memetic algorithm with novel semi-constructive evolution operators for permutation flowshop scheduling problem," *Applied Soft Computing*, vol. 94, p. 106458, 2020.

[43] M. Eusuff, K. Lansey, and F. Pasha, "Shuffled frog-leaping algorithm: a memetic meta-heuristic for discrete optimization," *Engineering Optimization*, vol. 38, no. 2, pp. 129–154, 2006.

[44] C. Sivamathi and S. Vijayarani, "mining high utility itemsets using shuffled complex evolution of particle swarm optimization (SCE-PSO) optimization algorithm," in *Proceedings of the 2017 International Conference On Inventive Computing And Informatics (ICICI)*, Coimbatore, India, November 2017.

[45] J. Zhang, C. Wang, and A. Zhang, "Shuffled complex evolution algorithm with opposition-based learning for a permutation flow shop scheduling problem," *International Journal of Computer Integrated Manufacturing*, vol. 28, no. 11, pp. 1220–1235, 2015.

[46] Y. Marinakis and M. Marinaki, "Particle swarm optimization with expanding neighborhood topology for the permutation flowshop scheduling problem," *Soft Computing*, vol. 17, no. 7, pp. 1159–1173, 2013.

[47] J. Li, Q. Pan, and S. Xie, "An effective shuffled frog-leaping algorithm for multi-objective flexible job shop scheduling problems," *Applied Mathematics and Computation*, vol. 218, no. 18, pp. 9353–9371, 2012.

[48] X. X. Zhang, Z. C. Ji, and Y. Wang, "An improved SFLA for flexible job shop scheduling problem considering energy consumption," *Modern Physics Letters B*, vol. 32, pp. 34–36, 2018.

[49] G. G. Samuel and C. C. Asir Rajan, "Hybrid: particle swarm optimization-genetic algorithm and particle swarm optimization-shuffled frog leaping algorithm for long-term generator maintenance scheduling," *International Journal of Electrical Power and Energy Systems*, vol. 65, pp. 432–442, 2015.

[50] Y. Zhang, X. X. Gao, Y. J. Wang et al., "Solving the time optimal traveling salesman problem based on hybrid shuffled frog leaping algorithm-genetic algorithm," *Journal of Electronics and Information Technology*, vol. 40, no. 2, pp. 363–370, 2018.

[51] L. Alhenak and M. Hosny, "Genetic-frog-leaping algorithm for text document clustering," *Computers, Materials & Continua*, vol. 61, no. 3, pp. 1045–1074, 2019.