Optimization Scheme for Flexible Job Shop Scheduling Considering Parallel Operations and Sequence Constraints of Jobs

Zhaohuan Zhuang[1,2], Yong Zhang[1,2]*

1. College of Electronics and Information Engineering, Shenzhen University, Shenzhen, 518060, China
2. Guangdong Provincial Key Laboratory of Intelligent Information Processing, Shenzhen, 518060, China

*Corresponding Author: yzhang@szu.edu.cn

Abstract. This paper investigates the Flexible Job Shop Scheduling Problem with Parallel Operations and Job Priority Constraints (JSPCPOJP). To address the limitations of existing job shop scheduling models that fail to accurately describe this type of problem, a mixed-integer programming-based optimization model is constructed. An improved genetic algorithm (RPEGA) is proposed to effectively solve the problem. The algorithm employs a Mixed Sequence and Operation Selection (MSOS) encoding method, a deep recursive population initialization mechanism, and POXIC crossover and EIB mutation operators that satisfy the problem constraints. It also combines elitism and roulette wheel selection strategies. The algorithm is validated through a simulation environment. The results show that the RPEGA algorithm can generate feasible solutions that meet the constraints of parallel operations and job priorities, significantly reducing the solution time. It demonstrates high solution efficiency and practicality, providing an effective method for solving similar scheduling problems in actual production.

Keywords: Flexible Job Shop Scheduling; Parallel Operations; Job Priority Constraints; Genetic Algorithm; Deep Recursive Population Initialization

1 Introduction

In the process of production in an engineering workshop, the following two situations may sometimes occur. First, some operations of different jobs need to be combined and processed in parallel. For example, a single cutting machine allows the simultaneous cutting of two jobs in one cutting task. Compared with the serial execution of tasks, this method can significantly reduce the total execution time of the cutting operations. Second, there are precedence constraints between different jobs. For example, in the production of an engineering sign, the processing of the sign base and the bending plate is required. The processing of the bending plate needs the base as a raw material.

Therefore, the base must be processed before the bending plate begins processing. From a priority perspective, the priority of the base job is higher than that of the bending plate job. The parallel cutting operations and the priority order between the base and the bending plate break the constraints of the job shop scheduling problem that a machine cannot process multiple different jobs at the same time and that jobs have the same priority. The introduction of this constraint significantly increases the complexity of intelligent workshop scheduling optimization. Among them, how to effectively deal with the collaborative optimization problem of parallel operations while satisfying the technological sequence constraints of jobs has become a technical challenge that urgently needs to be solved in the field of engineering workshop production scheduling. Therefore, this study defines it as Flexible Job Shop Scheduling Considering Parallel Operations and Job Priorities (JSPCPOJP).

2 Related Work

Domestic and international scholars have formed a relatively complete theoretical system in the research of job shop scheduling problems. From the perspective of model constraints, researchers have mainly considered machine availability, transportation time, transportation resources, and operation waiting time [1]. Although the above constraints increase the complexity of the construction and optimization of the Job Shop Scheduling (JSP) model, they significantly improve the model's fit to real production scenarios. With the deepening of research, scholars have expanded the Flexible Job Shop Scheduling Problem (FJSP) by introducing technological flexibility (such as alternative machines, variable process routes, and non-fixed operation sequences). However, it should be pointed out that whether it is the classical JSP or the FJSP, their core assumptions are still limited to the constraints of "a single job monopolizing machine resources" and "jobs having no correlation with each other [2]," and they do not cover the new constraint combination of parallel operation collaboration and job priority coupling that this study focuses on. Therefore, the consideration of parallel operations and job precedence constraints in this study is an important supplement to the existing JSP and FJSP research.

The research journey of optimization algorithms for the Flexible Job Shop Scheduling Problem (FJSP) can be primarily divided into the following two stages. Early Exploration Stage (1970s to 1990s): FJSP was first explicitly proposed by Brucker and Schlie in 1990 [3]. Prior to that, in 1977, Panwalkar and Lskander [4] introduced 113 allocation rules, which laid the foundation for subsequent research on scheduling problems. At that time, traditional exact algorithms such as branch and bound, cutting plane methods, and integer linear programming were widely used to solve FJSP. However, as the problem size increased, the computational complexity rose sharply, and the solution time failed to meet practical requirements. Heuristic and Metaheuristic Algorithm Emergence Stage (1990s to Early 21st Century): Metaheuristic algorithms such as Genetic Algorithm (GA), Particle Swarm Optimization (PSO), and Ant Colony Optimization (ACO) were gradually introduced [5]. These algorithms, with their wide search range, strong parallelism, and low

dependence on problem models, were able to effectively address the challenges of solving large-scale FJSP.

Genetic algorithms possess a number of significant advantages, such as parallelism, flexibility, adaptability, and scalability. When solving the FJSP or the JSP, they can demonstrate relatively excellent performance. Therefore, many scholars currently tend to improve and optimize genetic algorithms as a basis for their research on the (F)JSP [6]. The solution mechanism of genetic algorithms is based on the genetic encoding of individuals. This encoding method can achieve flexible modeling and expression of the problem. Based on this, researchers can design and define appropriate encoding schemes and corresponding operational procedures according to the specific requirements of the actual problem, thereby more accurately characterizing the problem's constraints and objective functions. Moreover, the scalability of genetic algorithms allows them to be combined with various other optimization methods to form hybrid optimization algorithms, which can further enhance the efficiency and effectiveness of solving problems. Considering the complexity of the JSPCPOJP and the potential demonstrated by genetic algorithms in solving similar problems, this paper proposes to conduct an in-depth study on the JSPCPOJP based on genetic algorithms, with the aim of achieving efficient solutions to this problem.

3 JSPCPOJP Model

3.1 Problem Description

The JSPCPOJP problem can be described as follows: Given N jobs to be processed on M machines, each job has a fixed processing route. However, multiple operations from different jobs can be combined into parallel operations and processed simultaneously on the same machine. Additionally, there are precedence constraints across jobs. To better reflect the actual production requirements, the problem also considers machine flexibility, meaning that each operation can be processed on multiple alternative machines, and the processing time depends on the selected machine. The scheduling objective is to assign appropriate machines to each operation and determine the optimal processing sequence, while satisfying machine availability, operation sequence constraints, parallel operation coordination constraints, and precedence constraints across jobs, in order to optimize the system performance metrics (such as the makespan). Table 1 provides a specific example with 4 jobs and 4 machines, where the numbers below the machines indicate the processing times of the corresponding operations on those machines.

Table 1. Example of JSPCPOJP

Workpiece	Pre Workpiece	Working Procedure	Optional Working Machine			
			M1	M2	M3	M4
J1	J2	O11	3	—	—	5
		O12	—	2	4	—
J2	—	O21	—	—	4	2
		O23	2	—	1	—
J3	—	O31	—	5	—	7
		O32	6	—	—	9
		O34	—	—	5	6
J4	—	O41	6	5	7	—
		O43	—	2	—	4
J5	—	O51	3	—	5	—
		O52	—	5	8	—
J6	—	O61	2	4	—	—
		O62	—	1	—	3
J2、J3	—	O22、O33	8	—	12	—
J2、J4	—	O24、O42	—	15	—	12

J1 represents job 1, O12 represents the second process of J1, and others are similar; J2 and J3 indicate that there are parallel execution processes in job 2 and job 3, and the corresponding processes are O22 and O33. The others are similar.

3.2 Mathematical Model

Based on the mixed-integer programming method [7], an optimization model is constructed with the objective of minimizing the maximum completion time, and its formal definition refers to the notation system shown in Table 2.

Objective Function:

$$C_{max} = min\big(max(C_m)\big) \quad 1 \le m \le M \tag{1}$$

Constraints:

$$S_{IJ} + X_{IJm}P_{IJm} \le C_{IJ} \quad \forall I, J, m \tag{2}$$

$$\sum_{m=1}^{M} X_{IJm} = 1 \quad \forall I, J \tag{3}$$

$$C_{IJ} \le C_{max} \quad \forall I, J \tag{4}$$

$$C_{IJ} + Q\big(Y_{IJPKm} - 1\big) \le S_{PK} \quad \forall I, J, P, K, m \tag{5}$$

$$C_{i(j-1)} \le S_U \quad \forall I, J, O_{ij} \in O_U \tag{6}$$

$$S_{IJ} \ge 0, P_{IJm} > 0, C_{IJ} > 0 \quad \forall I, J, m \tag{7}$$

$$S_{ij} \times X_{IJm} = S_{pq} \times X_{IJm} \quad \forall O_{ij}, O_{pq} \in O_{IJ} \tag{8}$$

$$C_{ij} \times X_{IJm} = C_{pq} \times X_{IJm} \quad \forall O_{ij}, O_{pq} \in O_{IJ} \tag{9}$$

$$S_{ij} \geq C_{i'j}, \quad \forall i, j \tag{10}$$

The mathematical model established in this study takes the parallel operation OIJ as the basic modeling unit, where the specific composition element Oij corresponds to the actual job operation. The model aims to minimize the maximum completion time (Equation 1) and includes the following key constraints: First, it ensures that each operation is processed on only one machine (Equation 3) and meets the rationality requirements of operation sequence, limiting the start time ≤ completion time (Equation 2); The maximum completion time is defined as the maximum value among the completion times of all operations (Equation 4); Second, it ensures that a machine can only process one operation at the same time (Equation 5). In response to the characteristics of the problem, the model innovatively designs the process coupling constraint (Equation 6), requiring that an operation must start after its predecessor operation in the same job is completed, and the parallel operation must wait for all related predecessor operations of its composition elements to be completed before it can start; It also strictly stipulates that the elements within the parallel operation must start and end synchronously (Equations 8-9). In addition, the model also considers the cross-job priority relationship constraint (Equation 10), requiring that the start time of an operation cannot be earlier than the completion time of its predecessor operation in another job, and through the non-negative constraint (Equation 7) to ensure the physical meaning of all time parameters.

Table 2. Explanation of Symbols

symbol	explain	symbol	explain
N	Number of pieces	$C_{IJ}(C_{ij})$	Completion time of $O_{IJ}(O_{ij})$
M	Number of machines	E_{IJm}	Completion time of O_{IJ} in machine m
I,P	Job number	P_{IJm}	Operation time of O_{IJ} in machine m
J,K	Operation set number	M_{IJ}	O_{IJ} optional machine set
m	Machine number	C_m	Completion time of the last process on machine m
O_{IJ}	Parallel operation $O_{IJ} = \{O_{ij}, O_{pq}, \dots, O_{yz}\}$	Q	An Infinite Real Number
N_i	Number of operations of job i	X_{IJm}	Whether O_{IJ} works on machine m is 1, otherwise it is 0
O_{ij}	Process j of job i,j ∈ $[1, N_i]$	Y_{IJPKm}	Whether O_{IJ} precedes O_{PK} on machine m is 1, otherwise it is 0
$S_{IJ}(S_{ij})$	Start time of $O_{IJ}(O_{ij})$	$C_{i'j}$	The maximum completion time of the preceding job i' of job i

4 Algorithm Design

To effectively address the scheduling optimization problem with parallel operations and cross-job precedence constraints, this study innovatively proposes the RPEGA algorithm model.

4.1 Encoding and Decoding Design

This study employs the MSOS (Mixed Sequence and Operation Selection) [8] encoding method to encode the JSPCPOJP problem. Given that the JSPCPOJP problem mainly consists of two subproblems, namely Operation Sequencing (OS) and Machine Assignment (MS), the chromosome is divided into two parts using the MSOS encoding. The first part is the Operation Sequencing (OS) section, as shown in Fig. 1, which represents the processing order of the operations. The second part is the Machine Assignment (MS) section, which specifies the machines allocated to each operation.

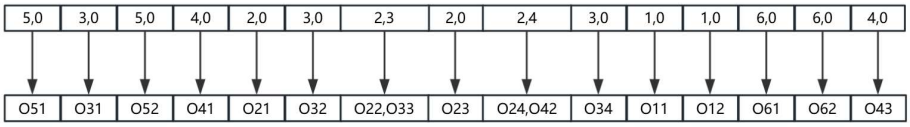

Fig. 1. Operation Encoding Sequence

When encoding the JSPCPOJP problem, each gene in the operation sequencing part is encoded using the index of the job set. The single operation and parallel operations are both represented by O_{ij}.To ensure consistency in the length of all genes, the number of elements in each gene is set to the maximum number of parallel operations, that is, $l = \max\{|O_{ij}|\}$, where i and j represent the indices of the job and operation, respectively. For cases where $|O_{ij}| < l$, the gene is padded with zeros to maintain uniform length, as shown in the first row of Fig. 1. In each gene, the frequency of a nonzero element i(with i ranging from [1, N]) appearing j times (with j ranging from [1, N_i]) represents the operation O_{ij} corresponding to job i, as shown in the second row of Fig. 1. In Fig. 1, the first "2,0" represents the single operation of Job 2, where "0" merely serves as a placeholder and has no practical significance. The notation "2,3" indicates a parallel operation between Job 2 and Job 3. Since "2" appears for the second time and "3" appears for the third time, "2,3" represents the parallel operation of the second operation of Job 2 and the third operation of Job 3.

Each gene in the machine code of the machine selection section is arranged from small to large according to the job and process, and its value x represents the xth machine among the available machines.

In the process of chromosome decoding, the encoded chromosome is first inputted, and the latest processing time (mp and jp) for each machine and each job is initialized to 0. Subsequently, each gene in the chromosome is traversed to explain the process represented by each gene. For example, gene "4,0" represents process 1 of job 4, while "2,5" indicates that process 2 of job 2 and process 1 of job 5 are parallel operations.

Next, obtain the machine selection and execution time corresponding to each process, for example, the machine selection for "4,0" may be 1, and the execution time may be 1. Then, take the larger of the latest machining time (mp) of the current machine and the latest machining time (jp) of the current job as the starting time (startTime) of the process, and calculate the finishing time of the process as startTime plus execution time (time). Finally, update the latest processing time of the current machine and job to the completion time of the process. After the completion of gene traversal, the latest completion time among all processes is the total completion time of the entire production plan.

4.2 Deep Recursive Population Initialization Method

In the process of initializing process encoding for FJSP, it is only necessary to create a sequence that includes all job processes and randomly shuffle their order. If this method is directly applied to the JSPCPOJP problem, the generated chromosomes cannot meet the constraints of job priority and parallel operations. For this purpose, this study designed a deep recursive population initialization method. During the initialization of the chromosome operation encoding, an empty chromosome is first created, and the initial queue of jobs that can be selected is determined based on the job priority constraints. For example, given the job priority constraints ["1;2,3", "3;4,5", "5;6,8", "7;5,9"], job 1 can only start processing after jobs 2 and 3 are completed. Therefore, initially, only jobs 2, 4, 6, 8, and 9 can be selected to be added to the chromosome. Next, a job is randomly selected from the queue_choose queue, and it is determined whether its current operation is a parallel operation. If the current operation is not a parallel operation, the operation is directly added to the chromosome. For example, adding operation 1 of job 2 results in adding "2,0" to the chromosome. If the current operation is a parallel operation, the other jobs in the parallel combination need to be iterated. If any of the other jobs in the combination are not in the queue_choose queue (i.e., their higher-priority jobs are not completed), no operation is added this time. If all other jobs in the combination are in the queue_choose queue, the status of each job is further checked. If the job has not yet reached the parallel operation stage, its non-parallel operation is directly added; if the job has reached the parallel operation stage but it is not the parallel operation to be executed currently, the job is used as input to jump to the "insert parallel operation into chromosome" procedure. If all jobs have reached the parallel operation stage, the parallel operation is added to the chromosome. Once an operation is added to the chromosome, whether it is a parallel or non-parallel operation, if all operations of the job are completed, the job is removed from the queue_choose queue. At the same time, it is checked whether other jobs belonging to the same priority constraint condition have also been completed. If they have all been completed, the subsequent jobs of this job are added to the queue_choose queue. The initialization process is shown in Fig. 2.

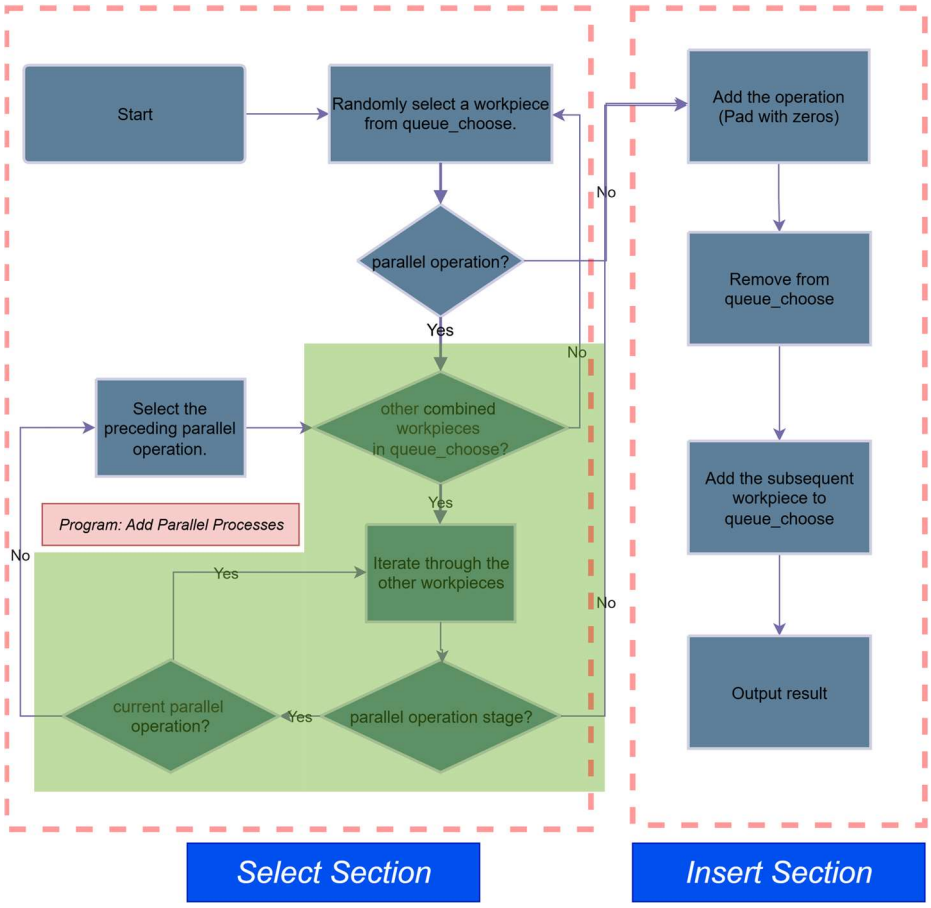

Fig. 2. Flowchart of Depth-First Recursive Initialization

4.3 Population Iteration Strategy

During the iterative process of the genetic algorithm, the key genetic operations involved include crossover, mutation, selection, and elitism [9]. For each generation of chromosomes, the mutation rate and crossover rate of the population are set in a segmented manner. The population is divided into three segments based on the fitness values: the top 20%, the middle 60% (20%-80%), and the bottom 20%. Segments with higher fitness values are assigned lower crossover and mutation rates to ensure the retention of high-quality individuals, while segments with lower fitness values are assigned higher crossover and mutation rates to ensure population diversity.

POXIC Crossover Operator.

In genetic algorithms, crossover operations are a core component for generating offspring with superior genetic combinations. Common crossover methods include

single-point crossover, multi-point crossover, uniform crossover, partially mapped crossover, operation sequence crossover, and extended sequence crossover [10]. However, these traditional crossover methods have limitations when dealing with parallel operations on the same machine, as they cannot effectively handle the coupling effects that arise, potentially leading to infeasible solutions in the offspring. To address this issue, a new crossover mechanism needs to be designed that satisfies the constraints of JSPCPOJP, ensuring the effectiveness and feasibility of the crossover operation. Based on the POX [11], this study proposes an improved crossover operator called POXIC (Precedence Operation Crossover Initiative Choose). The crossover principle is as follows: 1.Select one parent chromosome and identify the jobs that participate in the parallel constraints and job priority constraints. 2. Ensure that these jobs maintain the same processing order in the offspring chromosome as in the parent. 3. Select the remaining jobs from the other parent chromosome and fill them into the available positions in the offspring chromosome according to their processing order in the parent, thereby generating a new offspring chromosome. The advantage of this approach is that it ensures that subsequent jobs are not moved ahead of their predecessors and that parallel operations are executed within the specified number of operations. As shown in Fig. 3, jobs 1 and 2 have priority constraints, and jobs 2, 3, and 4 have parallel operation constraints. Therefore, jobs 1, 2, 3, and 4 are retained in their original positions from Parent 1 (P1) and added to the offspring chromosome (CP). Jobs 5 and 6 are then added to CP in the order they appear in Parent 2 (P2).

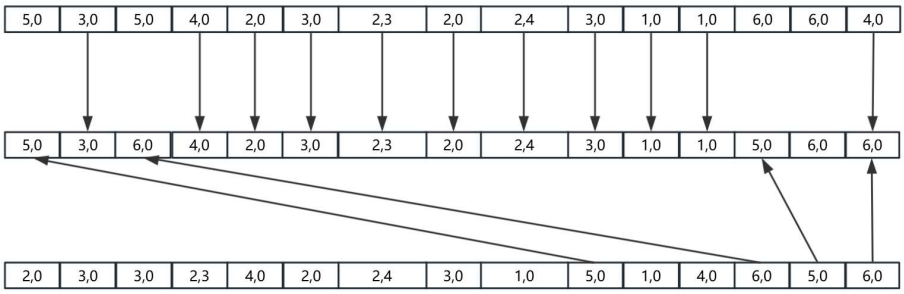

Fig. 3. POXIC Crossover Operator

The crossover operation for the machine encoding sequence is performed using a two-point crossover method [12].

EIB Mutation Operator.

Mutation plays a crucial role in genetic algorithms by introducing new genetic diversity to prevent premature convergence and enhance the algorithm's local search capability. When solving the (F)JSP problem, common mutation operations include swap mutation, inversion mutation, insertion mutation, and single-point mutation [13]. However, when these mutation operations are applied to the problem in this study, that is, the mutation of operation sequences, their inherent randomness may disrupt the job

priority constraints and parallel operation constraints. To ensure that the mutated operations still meet the constraints, this study proposes a boundary-based swap mutation operator, EIB (Exchange within the boundary). The implementation steps are as follows: a) Determine the swap boundary: Traverse the operation genes and set the gene positions of parallel operation x and the last operation y of the last job in the job priority constraint as the swap boundary. For operation x, both sides of its gene are set as boundaries, while for operation y, only the right side of its gene is set as a boundary. b) Divide the chromosome into n sub-chromosomes based on the boundaries. c) Randomly select a sub-chromosome with more than 2 genes from the n sub-chromosomes and randomly choose two gene positions within this sub-chromosome for swapping. As shown in Fig. 4, the genes (2,3) and (2,4) are parallel operations, and their left and right boundaries are set as swap boundaries. The gene (2,4) is the last operation of Job 2 and is the last operation in the job priority constraint 1>2, so its right boundary is set as a swap boundary. After determining the boundaries, the genes (4,0) and (3,0) in the first sub-chromosome are randomly selected for swapping, completing the mutation.

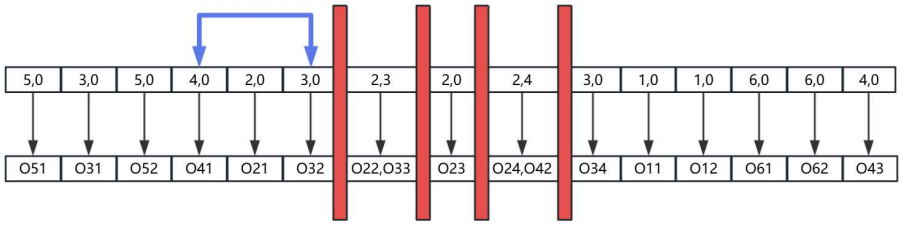

Fig. 4. EIB Mutation Operator

When implementing mutation operations on the machine encoding sequence, this study adopts a single-point mutation strategy.

The selection and retention strategy of the population.

The selection of the population during the iterative process employs a combination of elitism and roulette wheel selection strategies [14]. The specific procedure is as follows: a) The new chromosomes generated by crossover and mutation in each generation are added to the population of the previous generation. b) The population, now including the new chromosomes, is sorted in descending order based on the fitness values of each chromosome, and the fitness values of all individuals are summed. c) The cumulative fitness value of each individual is divided by the total fitness value to obtain the probability interval for each individual. d) The top 20% of chromosomes with the highest fitness values are directly retained as elites for the next generation. e) For the remaining 80% of chromosomes, a random number between (0,1) is generated each time. The individual whose probability interval the random number falls into is selected and added to the new population.

5 Experiment

5.1 Simulation Environment

To verify the effectiveness of the improved genetic algorithm proposed in this study, a simulation environment needs to be established to solve the JSPCPOJP problem under the corresponding parameters using the improved genetic algorithm designed in this study. The scheduling program was constructed using Java in this study, and all experiments were completed on a personal computer with the Windows 10 operating system, an Intel Core i7-8550U CPU, and 8GB of memory.

5.2 Effectiveness Validation

The scheduling was performed for the case in Table 1, with the population size set to 100 and the maximum number of iterations set to 50. The crossover rates for the top 20%, middle 40% (20%–60%), and bottom 20% of the population were set to 0.5, 0.7, and 0.9, respectively. The mutation rates for the top 20%, middle 40% (20%–60%), and bottom 20% of the population were set to 0.1, 0.3, and 0.5, respectively. TheGantt chart output is shown in Fig. 5:

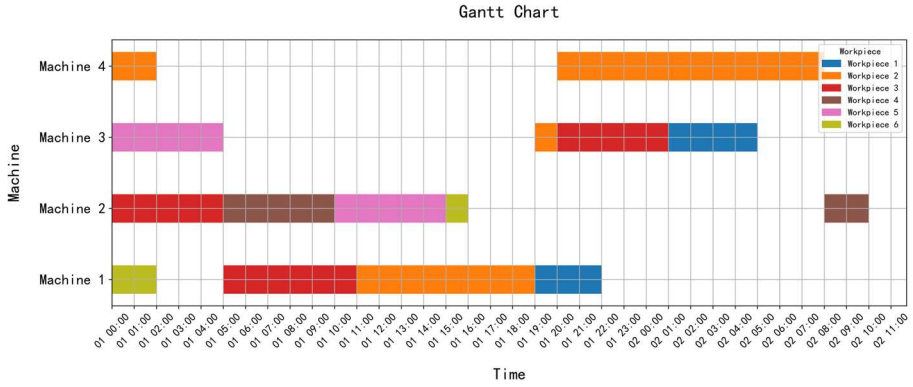

Fig. 5. Gantt Chart for Scheduling of Case 1

The inspection of the scheduling results shows that: Job 1 starts processing after Job 2 is completed. Before the parallel operations O22_O33 are processed, their preceding operations O21, O31, and O32 have been completed. Before the parallel operations O24_O42 are processed, their preceding operations O21, O22_O33, O23, and O41 have been completed. The scheduling results satisfy the job priority constraints and the parallel operation constraints.

5.3 Comparison with Exact Solution Methods

Based on the international benchmark instances of SFJS10 and MFJS1-4 [15], test instances POJPM1-10 were generated by randomly introducing one or two parallel

operations and one or two priority constraints. The specifications of these instances are shown in Table 3:

Table3. POJPM Instances

Test Case	Basic Case	Workpiece Priority	Parallel Operations	Available Machines	Operation Time
POJPM1	SFJS10	1>2	O22,O33	M1/M2/M4	22/21/24
POJPM2	SFJS10	1>2	O21,O31	M1/M2/M4	22/21/24
		1>3	O22,O33	M1/M2/M4	22/21/24
POJPM3	MFJS1	1>2	O22,O33	M1/M3	14/13
POJPM4	MFJS1	1>2	O21,O31	M2/M3/M4	21/20/22
		1>3	O22,O33	M1/M3	14/13
POJPM5	MFJS2	1>2	O22,O33	M1/M3	14/13
POJPM6	MFJS2	1>2	O21,O31	M2/M3/M4	21/20/22
		1>3	O22,O33	M1/M3	14/13
POJPM7	MFJS3	1>2	O22,O33	M1/M3	14/13
POJPM8	MFJS3	1>2	O21,O31	M4	22
		1>3	O22,O33	M1/M3	14/13
POJPM9	MFJS4	1>2	O22,O33	M1/M3	14/13
POJPM10	MFJS4	1>2	O21,O31	M4	22
		1>3	O22,O33	M1/M3	14/13

The exact optimal solutions for each case were obtained by traversing all possible solutions, and the solution time was recorded. Meanwhile, the approximate solutions for each case were obtained using RPEGA algorithm, and the solution time was also recorded. The results are shown in Table 4: where J represents the number of jobs in the instance, O represents the maximum number of operations per job, M represents the number of optional machines, Cmax1 represents the minimum makespan obtained by the exact solution method, Cmax2 represents the minimum makespan obtained by the RPEGA algorithm, CPUs1 represents the solution time of the exact solution method, and CPUs2 represents the solution time of the RPEGA algorithm. The unit of CPUs is milliseconds, and the unit of Cmax is hours.

Table4. Comparison Results between Exact Solution Method and RPEGA Algorithm

Test Case	J x O x M	Cmax1	CPUs1	Cmax2	CPUs2
POJPM1	4 x 3 x 5	67	14064	67	4408
POJPM2	4 x 3 x 5	75	6018	75	3617
POJPM3	5 x 3 x 6	43	19189	43	4492
POJPM4	5 x 3 x 6	46	6140	46	3974
POJPM5	5 x 3 x 7	43	30451	43	4767
POJPM6	5 x 3 x 7	46	9193	46	4079
POJPM7	6 x 3 x 7	48	42659	48	5036
POJPM8	6 x 3 x 7	48	10390	48	3675
POJPM9	7 x 3 x 7	48	343193	48	5282
POJPM10	7 x 3 x 7	48	93655	48	4233

It can be seen that our algorithm is able to obtain feasible solutions for these small-scale JSPCPOJP instances while significantly reducing the solution time, thereby demonstrating the feasibility of the model. Below is an analysis of the time complexity

of our algorithm: Let N be the population size, L represent the number of bits in a chromosome, P denote the number of parallel operations in a chromosome, b be the average number of jobs in parallel operations, S represent the number of job priority constraints in a chromosome, and c be the average number of jobs in each job priority constraint.In the worst case, the time complexities for the operations of initialization, crossover, mutation, and elitism are O(N×((L-P-S)+Pb+Sc), O(N×L), O(N×L), and O(log(N)+O(N)), respectively. Considering that in most cases, P, S, b, and c are smaller than L, the time complexities can be simplified to O(N×L), O(N×2L), O(N×L), and O(N), respectively. The overall time complexity is O(N×L).From this analysis, it can be inferred that the time complexity of our algorithm is not high, which accounts for its faster execution speed.

5.4 Algorithm Comparison

In order to compare the solution performance of RPEGA in large-scale instances, a case with dimensions of 20×14×10 was designed in this section. The maximum number of selectable machines for each operation was set to 2, and three job priority constraints as well as two parallel operation constraints were added. Subsequently, the instance was solved using the initial GA model (without the crossover and mutation segmentation probability operators), the genetic annealing model (SA), and the proposed RPEGA model, respectively. The solution iteration process is illustrated in Fig. 6:

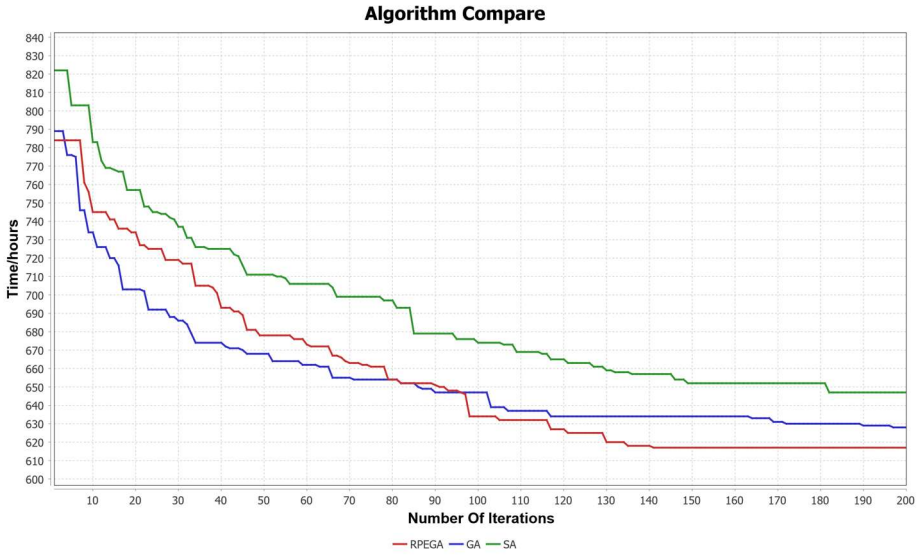

Fig. 6. Algorithm Comparison

It can be seen that in large-scale use cases, the RPEGA model can obtain better solutions within a limited number of iterations

6 Conclusion

This paper addresses the Flexible Job Shop Scheduling Problem with Parallel Operations and Job Priority Constraints (JSPCPOJP) by constructing a corresponding optimization model and proposing an improved genetic algorithm (RPEGA). By introducing the Mixed Sequence and Operation Selection (MSOS) encoding method, a deep recursive population initialization mechanism, and improved crossover and mutation operators, the algorithm can effectively handle parallel operations and job priority constraints, generating feasible scheduling solutions that meet the constraints. Experimental results show that the RPEGA algorithm can quickly obtain approximate solutions close to the exact optimal solutions when solving small-scale JSPCPOJP problems, with significantly less solution time compared to exact solution methods. In the comparative tests on large-scale instances, the RPEGA algorithm obtained better solutions than the initial genetic algorithm and the genetic annealing algorithm within limited a number of iterations. This indicates that the RPEGA algorithm possesses superior global search capabilities and convergence speed when dealing with large-scale problems, and is capable of effectively addressing complex scheduling constraints. Future research can further explore the optimization potential of the algorithm to address more complex scheduling scenarios and larger-scale problem instances.

Acknowledgments. This work was supported by Guangdong Provincial Key Laboratory（Grant 2023B1212060076 ） and Shenzhen Science & Technology Plan (No. KJZD20230923114405012).

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

References

1. Dauzère-Pérès S, Ding J, Shen L, et al. The flexible job shop scheduling problem: A review[J]. European Journal of Operational Research, 2024, 314(2): 409-432.
2. Xie, J., Gao, L., Peng, K., Li, X., & Li, H. (2019). Review on flexible job shop scheduling. IET collaborative intelligent manufacturing, 1(3), 67-77.
3. Brucker P. Schlie R. Job-shop scheduling with multi-purpose machines[J]. Computing, 1990, 45(4): 369-375.
4. Panwalker S, Wafik Lskander. A survey of scheduling rules[J]. Ops.Res., 1977, 25(1):45-61.
5. Chaudhry I A, Khan A A. A research survey: review of flexible job shop scheduling techniques[J]. International Transactions in Operational Research, 2016, 23(3): 551-591.
6. Amjad M K, Butt S I, Kousar R, et al. Recent research trends in genetic algorithm based flexible job shop scheduling problems[J]. Mathematical Problems in Engineering, 2018, 2018(1): 9270802.

7. Fattahi, P., Saidi Mehrabad, M., & Jolai, F. (2007). Mathematical modeling and heuristic approaches to flexible job shop scheduling problems. Journal of intelligent manufacturing, 18, 331-342.
8. Ho N B, Tay J C. GENACE: an efficient cultural algorithm for solving the flexible job-shop problem[C]//Proceedings of the 2004 Congress on evolutionary computation (IEEE Cat. No. 04TH8753). IEEE, 2004, 2: 1759-1766.
9. Cheng R, Gen M, Tsujimura Y. A tutorial survey of job-shop scheduling problems using genetic algorithms, part II: hybrid genetic search strategies[J]. Computers & Industrial Engineering, 1999, 36(2): 343-364.
10. Davis L. Job shop scheduling with genetic algorithms[C]//Proceedings of the first International Conference on Genetic Algorithms and their Applications. Psychology Press, 2014: 136-140.
11. Zhang C, Rao Y, Li P. An effective hybrid genetic algorithm for the job shop scheduling problem[J]. The International Journal of Advanced Manufacturing Technology, 2008, 39: 965-974.
12. Watanabe M, Ida K, Gen M. A genetic algorithm with modified crossover operator and search area adaptation for the job-shop scheduling problem[J]. Computers & Industrial Engineering, 2005, 48(4): 743-752.
13. Li X, Guo X, Tang H, et al. Survey of integrated flexible job shop scheduling problems[J]. Computers & Industrial Engineering, 2022, 174: 108786.
14. Yu F, Fu X, Li H, et al. Improved roulette wheel selection-based genetic algorithm for TSP[C]//2016 International conference on network and information systems for computers (ICNISC). IEEE, 2016: 151-154.
15. Bagheri A, Zandieh M, Mahdavi I, et al. An artificial immune algorithm for the flexible job-shop scheduling problem[J]. Future Generation Computer Systems, 2010, 26(4): 533-541.