Mathematical Biosciences and Engineering

MBE, 20(9): 17407–17427.
DOI: 10.3934/mbe.2023774
Received: 25 May 2023
Revised: 20 August 2023
Accepted: 22 August 2023
Published: 11 September 2023

http://www.aimspress.com/journal/MBE

Research article

An improved genetic algorithm with dynamic neighborhood search for job shop scheduling problem

Kongfu Hu[1], Lei Wang[1], Jingcao Cai[1,2,*] and Long Cheng[1]

[1] School of Mechanical Engineering, Anhui Polytechnic University, Wuhu 241000, China
[2] AnHui Key Laboratory of Detection Technology and Energy Saving Devices, AnHui Polytechnic University, Wuhu 241000, China

* **Correspondence:** Emil: caijingcao@foxmail.com.

Abstract: The job shop scheduling problem (JSP) has consistently garnered significant attention. This paper introduces an improved genetic algorithm (IGA) with dynamic neighborhood search to tackle job shop scheduling problems with the objective of minimization the makespan. An inserted operation based on idle time is introduced during the decoding phase. An improved POX crossover operator is presented. A novel mutation operation is designed for searching neighborhood solutions. A new genetic recombination strategy based on a dynamic gene bank is provided. The elite retention strategy is presented. Several benchmarks are used to evaluate the algorithm's performance, and the computational results demonstrate that IGA delivers promising and competitive outcomes for the considered JSP.

Keywords: job shop scheduling problem; improved genetic algorithm; idle time; improved POX; neighborhood searching; dynamic gene bank; elite retention

1. Introduction

In the context of smart manufacturing and in line with the "Made in China 2025" strategy, enterprises strive for intelligent and precise production. Job shop scheduling is a control technology that can optimize production processes. By employ intelligent algorithms, we can efficiently utilize production resources despite existing constraints, leading to cost savings and naturally shortening the manufacturing cycle.

There are several types of job shop scheduling, including the Job shop scheduling problem (JSP), Flow shop scheduling problem (FSP), and others. The optimization objective can also be classified into the single objective and the multiple objective. JSP represents the most fundamental and representative production problem. In brief, it involves processing several jobs on machines, where the processing information is determined before machining starts, and the goal is to minimize the makespan.

Research on JSP holds both academic and practical significance, making it a continuously evolving and debated topic. The solution methods for JSP can be broadly categorized into precise methods and approximate methods. The exact method can determine the exact optimal solution for the problem, but as the problem's scale grows, its time complexity increases significantly, making it impractical in real-world scenarios. On the other hand, the approximate method may not guarantee an optimal solution, but it can provide a relatively optimal feasible solution within a reasonable amount of time. At present the research on JSP is focused on modern intelligent algorithms and their improved hybrid algorithms. These algorithms have been very successful in solving JSP, including genetic algorithm (GA) [1–3], tabu search [4,5], particle swarm optimization (PSO) [6,7], ant colony optimization [8,9], etc.

GA as one of the classical heuristics was first proposed by Holland, and it is a highly parallel adaptive intelligence algorithm that is inspired by Darwin's theory of evolution [10]. In contrast to alternative intelligent algorithms, the GA demonstrates a higher propensity for uncovering global optimal solutions. This proclivity arises from GA's reliance on an ensemble of candidate solutions, as opposed to a solitary solution. Moreover, within numerous instances, specific operations such as crossover and mutation engender discrepancies within the candidate solution relative to its antecedent. This dynamic characteristic serves to avert premature confinement within local optima. With the development of technology, traditional GA is no longer sufficient to solve all problems, so, many scholars have optimized different steps in GA such as coding [11–13], crossover [14,15], mutation [16,17], etc. and achieved certain results; Other scholars have also proposed some effective improvement strategies. Goncalves et al. [18] proposed a search strategy: the graphical method of Akers, combined with GA, it's a strange idea that he transmuted the JSP into the path planning problem, by using this method he optimized process arrangement. Zhang et al. [19] extend the N6 neighborhood and apply it to the Tabu search, their approach dominates others in terms of both solution quality and performance. Chen et al. [20] proposed a hybrid quantum algorithm based on a local optimization strategy and improved optimization of the rotation angle. Besides, some people also have incorporated reinforcement learning into their algorithms to solve JSP. Wang et al. [21] used the Q-learning algorithm to select appropriate scheduling rules. Shylo et al. [22] propose global equilibrium search method, which has some common features with the simulated annealing method. Nasiri et al. [23] used a scatter search algorithm combined with tabu search and path relinking for the partial JSP, they constructed new solutions by combining existing ones in a systematic fashion, and the computational experiments show that this algorithm is effective. Gui et al. [24] proposed an effective memetic algorithm (EMA) to solve the multi-objective JSP, a new hybrid crossover operator is designed to enhance the search ability of the proposed EMA and avoid premature convergence.

The majority of scholars aim to enhance algorithm performance by refining existing methods or introducing new strategies. The primary concern to address is enhancing the search capability and improving the quality of the approximate method solution within a given time frame. Nevertheless, a single improvement alone is insufficient. To tackle various issues and common drawbacks in the GA, we incorporate several beneficial enhancements. The goal is to enhance algorithm performance by improving existing methods or proposing new strategies, and the main problem that needs to be solved is enhancing the search capability and improving the quality of the approximate method solution within a given time. However, only one improvement is not enough. To address various issues and common disadvantages in GA, we introduce several useful enhancements. In the decoding process, we design an additional program to search idle time between processes on the machine and identify a suitable subsequent process that can be processed in advance. This program can improve the compactness of the solution. In response to the shortcomings of the common crossover operator, such as high randomness and low efficiency, a Precedence Operation Crossover (POX) operator is adopted and improved to retain favorable genes of the parents while minimizing the damage to the chromosome as much as possible. In the mutation operator, traditional methods cannot create excellent new genes efficiently. Therefore, we propose a new mutation operator with neighborhood optimization, it can effectively reduce ineffective inheritance. To further enhance the sustained searching ability of the algorithm, we introduce a new genetic recombination strategy with dynamic gene banks. This enables the algorithm to maintain relatively strong search ability even in later iterations. Simultaneously, we incorporate an elite retention strategy to prevent the loss of the best solutions in each iteration.

The paper is structured as follows. The subsequent section provides an introduction of JSP. Section 3 outlines the specific improvement steps applied to the algorithm. In Section 4, we present the results of the enhanced GA on benchmarking problems, along with the corresponding analysis. Finally, in Section 5, we provide our conclusions.

2. Description of job shop scheduling problem

There are n jobs processed on m machines in JSP. The set of jobs $J=\{J_1,J_2,...,J_n\}$ and the set of machines $M=\{M_1,M_2,...,M_m\}$. Each job consists of some operations and $O_{i,j}$ is the j-th operation of J_i. C_i represents the completion time of J_i. $T_{i,j}$ denotes the process time of $O_{i,j}$.

There are some other constrains as follows.

1) Every machine can process only one job at a time;

2) Throughout the manufacturing process, each job cannot be processed target times on the same machine.

3) The processing of the operations cannot be interrupted after it has begun;

4) There is no preparation time between all processes;

5) All machines and all jobs are available from time 0 on;

6) The operations of the same job should comply with the constraints of the process route.

7) Imposes the start time of all operations are non-negative.

The goal of JSP is to minimize the makespan: $C_{\max}$.

$$C_{\max}=\min(\max_{i=1}^{n} C_i) \tag{1}$$

A 4 × 4 case is given, which means this case has 4 jobs and 4 machines, and the information is shown in Table 1. Figure 1 illustrates a feasible schedule. generated randomly under these processing constraints.

Table 1. The information of a 4 × 4 case.

Job / *i*	Processing time/Machine			
	$O_{i,1}$	$O_{i,2}$	$O_{i,3}$	$O_{i,4}$
1	3/1	3/2	2/3	6/4
2	1/1	5/4	3/3	4/2
3	3/2	2/1	3/4	5/3
4	3/4	2/3	4/2	1/1

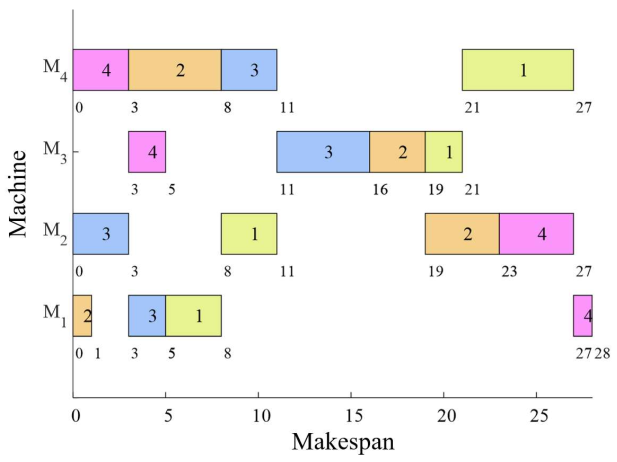


Figure 1. A Gantt chart of a 4×4 case.

The horizontal axis of the Gantt chart represents the makespan, while the vertical axis represents the processing machines. In the chart, we utilize the same color to represent different operations of the same job and distinguish different jobs with distinct colors. Each job's processing on a machine is represented by a rectangular block, with the job number indicated inside the rectangle. The numbers at both ends of the rectangle denote the start and finish time of the corresponding job processing.

3. The improved GA

3.1. The GA

The classical GA is an evolutionary algorithm that leverages the collaborative efforts of multiple individuals to achieve its impressive global search ability. The initialization of GA involves creating feasible solutions through specific methods, followed by the selection of valuable solutions to form a new population. To obtain better solutions and achieve population convergence, the crossover operator and mutation operator are commonly employed as search methods. The crossover operator functions as a global search method, where two individuals are selected from their respective parents and combined through hybridization to create a new offspring. On the other hand, the mutation operator is a local search method, it usually creates a new individual by adding a small disturbance to the individual's local area. Repeat the select operator, crossover operator and mutation operator mentioned above until the preset termination conditions are met. The flowchart for GA is illustrated in Figure 2.

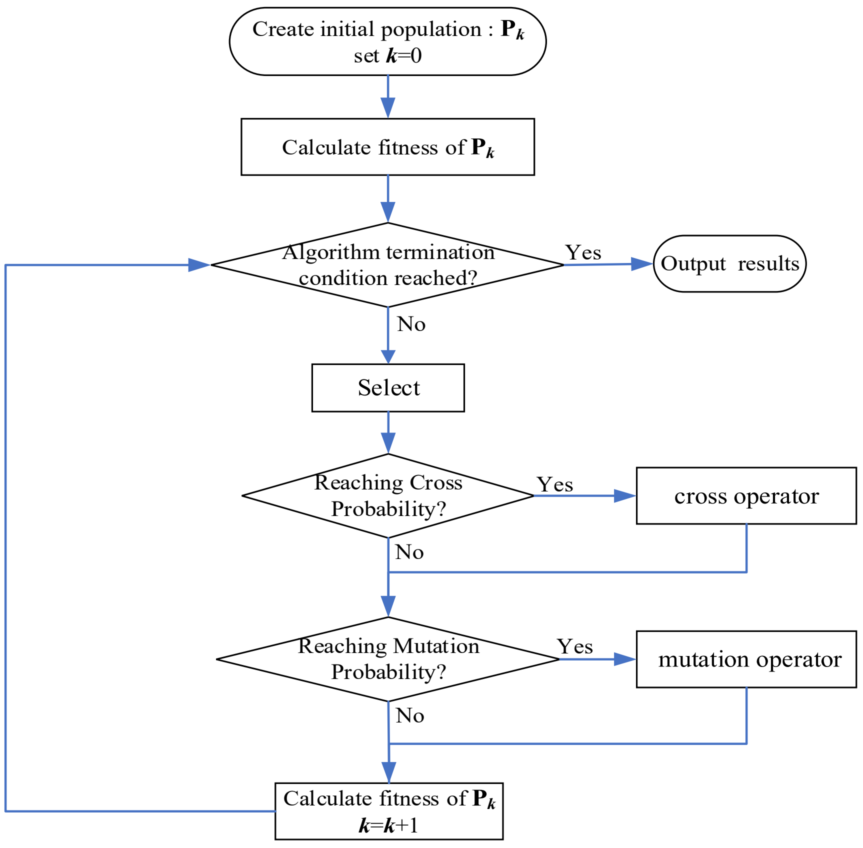

Create initial population : $\mathbf{P}_k$
set k=0
Calculate fitness of $\mathbf{P}_k$
Algorithm termination condition reached?
Yes
Output results
No
Select
Reaching Cross Probability?
Yes
cross operator
No
Reaching Mutation Probability?
Yes
mutation operator
No
Calculate fitness of $\mathbf{P}_k$
k=k+1

Figure 2. Flowchart for GA.

3.2. Encoding and decoding

Encoding constitutes the initial and pivotal phase of a GA, exerting a direct influence on both its complexity and precision. Various established encoding methods exist, encompassing process-based encoding, job-based encoding, machine-based encoding, and even hybrid encoding. These distinct encoding strategies serve to capture diverse attributes, with the selection to be made in alignment with specific requisites.

In this paper, a job-based encoding approach is adopted. The chief advantage of this method lies in its capacity to readily and straightforwardly identify both the job and its associated processes. Furthermore, the chromosome generated through this approach remains amenable to direct implementation across a spectrum of genetic operators, including crossover and mutation operators. The instantiation of initial chromosomes involves the application of random numbers. This approach ensures maximal genetic diversity within the population, consequently mitigating the risk of premature entrapment within local optima during subsequent genetic operations.

In the chromosome, the number indicates the job number, and the number of occurrences indicates the process of the job. In JSP, each job needs to be processed according to predetermined constraints. In the job-based encoding, the length of the chromosome is related to the total number of processes. Based on the 4 × 4 case we mentioned in Section 2 as an example. In this case, there are a total of 4 jobs and 4 machines, and each job has 4 processes. Therefore, the length of the chromosome is 16 bits. First, we generate a sufficiently long numerical encoding string in a specific order: [1111222233334444], and then shuffle it by using random numbers, it will become like this: [3243124313224114], a new chromosome has been created. The "3" in [3243...] indicates the job "J_3", the first "3" represents the first process of "J_3" ($O_{3,1}$), the second "3" represents the second process of "J_3" ($O_{3,2}$), and so on. The diagram of this coding method is shown in Figure 3.

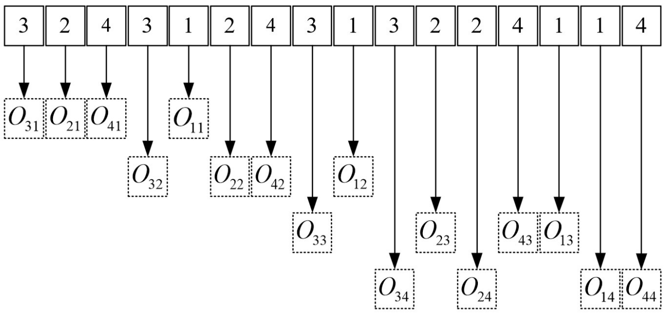

3 2 4 3 1 2 4 3 1 3 2 2 4 1 1 4
O_{31} O_{21} O_{41} O_{11}
O_{32} O_{22} O_{42} O_{12}
O_{33} O_{23} O_{43} O_{13}
O_{34} O_{24} O_{14} O_{44}

Figure 3. The diagram of a job-based encoding.

The usual method for decoding involves calculating the makespan based on the chromosome and processing information at the outset. Due to the random nature of chromosome creation, certain machines may exhibit relatively more idle time. To address this, a search for machine idle time is incorporated and coupled with process advancement. The fusion of these two approaches results in chromosome adjustments that effectively minimize idle time, thus enhancing the overall solution quality. The specific operational steps are outlined as follows:

Step1: Decode. Set a collection A to store the start and end times of each operation on each machine. In collection A, if the end time of one operation equals the start time of the next operation, it indicates that there is no idle time between the two operations. Otherwise, there is idle time present. Set collection B to record the idle time between operations on each machine, along with their start and end times. If there is no idle time, it will be represented as 0. The given processing information includes the processing time for each operation of every job, it will be stored in collection C.

Step2: Find and record every idle time on each machine.

Step3: Start from the first idle time on the first machine, and search for subsequent processes one after another following this idle time, if any process adheres to either of the following two conditions, proceed to Step 4; otherwise, proceed to Step 5.

Case 1: $a \le b, c \ge d$;

Case 2*:* $a > b; (c-a) \ge d$

where, a represents "the start-moment of the previous process of the current job", b represents "the idle time start-moment", c represents "the length of idle time", and d represents "the processing time of the current process". All the data above is read from collections A, B, and C.

Step4: dvance the process to the idle time for processing, and update the chromosome.

Step5: Continue searching for the next idle time and proceed to Step 3. If the current idle time is the last one, initiate a search for the next available machine and return to Step 3. In the event that all idle times on each machine have been exhaustively searched, and none of them satisfy the aforementioned two conditions, signifying the inability to utilize any idle time, the program is terminated.

A diagrammatic representation of the 2 types of idle time mentioned in Step3, see Figure 4.

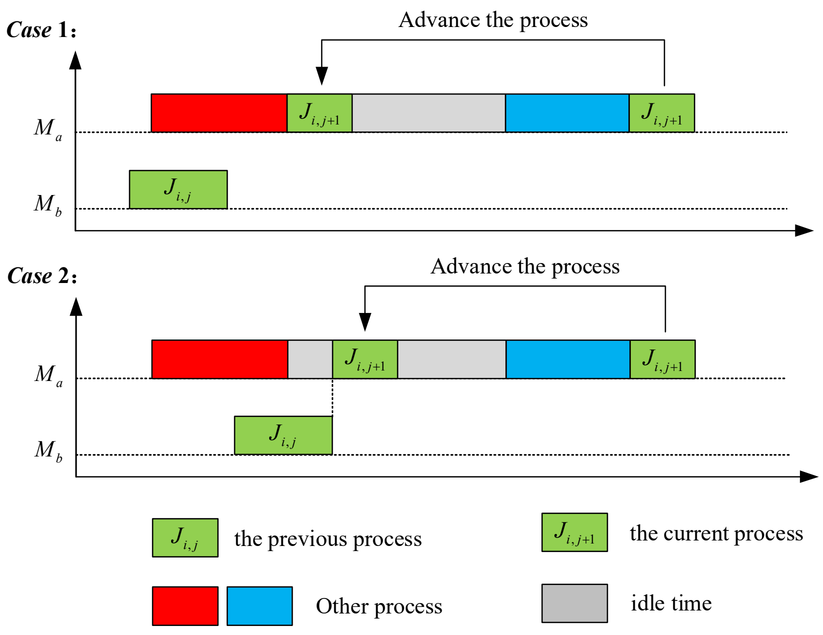


Figure 4. Two types of idle time and process advancing.

For example, the chromosome mentioned above [3243124313224114], we decode it and search for idle time, chromosomes will be adjusted to [3243124312324114], the new scheduling of the new chromosome is plotted as a Gantt chart in Figure 5. The completion time for this updated schedule has been notably reduced, from 28 seconds down to 24 seconds, this compelling outcome serves as evidence affirming the efficacy of the idle time search and utilization strategy.

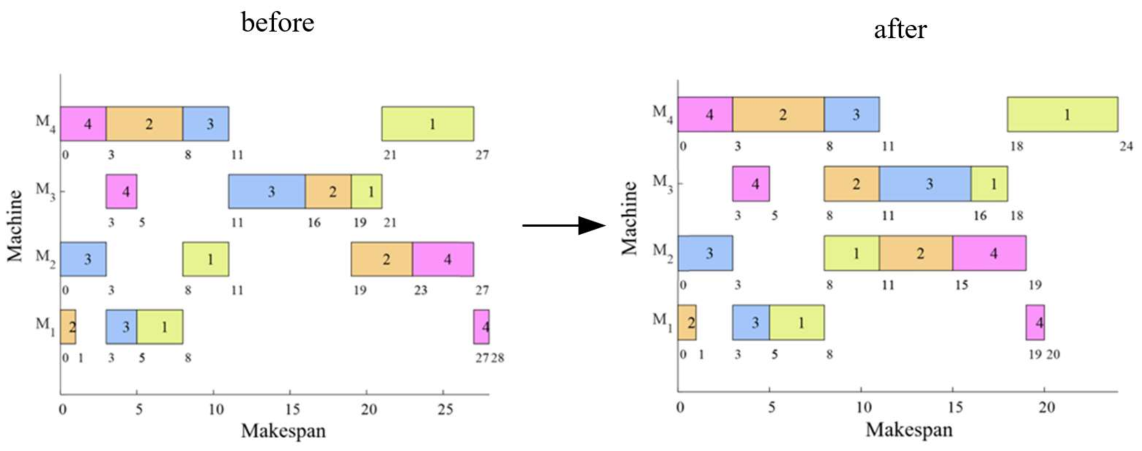


Figure 5. The Gantt chart before and after optimization.

3.3. Selection operator and elite retention

In IGA, not all chromosomes are allowed to enter the next generation population. In this paper, we employ the championship method to select chromosomes, referred to as high-quality individuals. Initially, two chromosomes are randomly chosen, and their fitness is compared, the superior one is retained in the new generation. This process is repeated until the new generation's size reaches the predetermined value.

To prevent the loss of the best chromosome in each generation and to enhance overall algorithm convergence, we introduce the elite retention strategy in IGA. The best chromosome of each generation is recorded in the selection operator, after a series of genetic operations, if the best solution in the current generation is inferior to that of the previous generation, the best solution from the previous generation replaces the worst solution of the current generation. By incorporating the championship method and elite retention, the IGA aims to maintain the best individuals over generations and improve the algorithm's overall performance and convergence.

3.4. An improved POX crossover operator

The crossover operator functions as a global search mechanism, with certain commonly employed variants including the single-point crossover operator, multi-point crossover, and mixed crossover operator, among others. While these traditional crossover methods possess inherent search capabilities, they also carry a certain risk of compromising high-quality genes. Addressing this concern, Zhang et al. introduced an innovative crossover approach known as the Precedence Operation (POX) crossover operator. This method adeptly inherits parental traits while mitigating the potential gene damage, consistently yielding feasible new solutions. Empirical evidence validates its superiority over alternative crossover techniques.

However, according to the description of the traditional POX crossover operator, it always selects consecutive job identifiers when partitioning the set of jobs. Operating in this manner, as the number of iterations increases, the offspring are likely to exhibit similar shortcomings, leading to a reduction in genetic diversity, which is unfavorable for the algorithm's convergence. Therefore, we propose an improvement to the POX crossover operator: when partitioning the set of jobs, we will use a random method in both quantity and individual aspects. This approach will help mitigate the impact caused by the aforementioned issues. The specific process of the improved POX crossover operator is as follows:

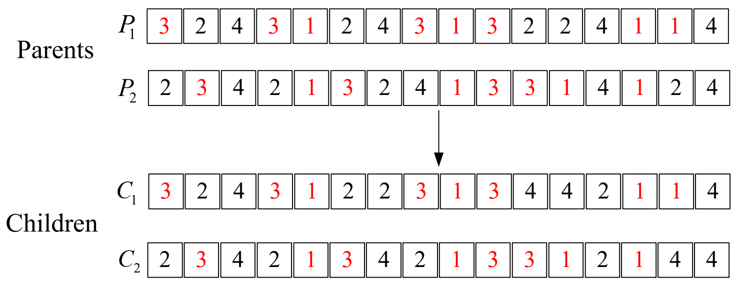
Step1: Set the size of the job set: $n_1 (1 < n_1 < n)$;

Step2: Select n_1 jobs into a collection: J_1, put other jobs into another collection: J_2;

Step3: Copy the individual from parent P_1 that contain jobs in J_1 to individual C_1 and preserve their positions. Similarly, copy the individual from parent P_2 that contain jobs in J_2 to collection C_2 and maintain their positions.

Step4: Copy the individual from parent P_2 that contain jobs in J_2 to collection C_1, preserving their order. Similarly, copy the individual from parent P_1 that contain job J_1 to collection C_2, also preserving their order.

Figure 6 illustrates the crossover operator of two parents in a 4 × 4 scheduling case. It can be observed that the improved POX crossover operator ensures that each offspring contains genes from both parents. Moreover, the sequence of job processing on machines is completely preserved and not disrupted during the crossover operator.



Parents
P_1 3 2 4 3 1 2 4 3 1 3 2 2 4 1 1 4
P_2 2 3 4 2 1 3 2 4 1 3 3 1 4 1 2 4
Children
C_1 3 2 4 3 1 2 2 3 1 3 4 4 2 1 1 4
C_2 2 3 4 2 1 3 4 2 1 3 3 1 2 1 4 4

Figure 6. An improved POX crossover operator.

3.5. A new neighbourhood seeking mutation operator

Mutation is a crucial local search ability in GA that generates new genes, enhancing the performance of offspring and preserving chromosome diversity. However, these traditional mutation operators often lack control and frequently fail to achieve steady progress. In fact, they may introduce inferior genes, leading to ineffective inheritance. For instance, when using single-point or multi-point inverse-order mutation operators, the randomness of the selected points can result in excessive perturbation to the chromosome, particularly when dealing with long selected parts, this hampers the evolution of individuals. Similarly, the process swap and insertion mutation operators cause minimal perturbation to the entire chromosome, becoming practically negligible as the problem size increases. As a consequence, the traditional GA suffers from low local search capability and exhibits poor overall performance. These limitations highlight the need for innovative mutation strategies to improve the algorithm's effectiveness.

We introduce an innovative mutation operator grounded in neighborhood exploration. To initiate this process, we randomly select a point within the chromosome, defining a neighborhood region (typically spanning around 10% of the chromosome's length). Subsequently, genes are randomly shuffled to generate a fresh chromosome configuration. Following this, the modified chromosome is decoded, and the resulting completion time is recorded. Should the completion time of the newly generated chromosome prove shorter than that of the original, the former supersedes the latter. Conversely, if the completion time is not improved, the original chromosome remains unchanged. This operation can be iterated multiple times to achieve refinement. The specific procedural steps are delineated as follows:

Step1: Set mutation times: k .Set $i = 0$.

Step2: Set a point randomly in the chromosome and form a neighbourhood.

Step3: Shuffle genes randomly in this neighbourhood.

Step4: Decode the new chromosome.

Step5: If the new one is better than the original, let the new one replaces the original; if not, remain unchanged.

Step6: $i = i + 1$.

Step7: If $i > k$, end; if not, go to Step2.

The diagram of the mutation operator in Steps2 and 3 is shown in Figure 7.

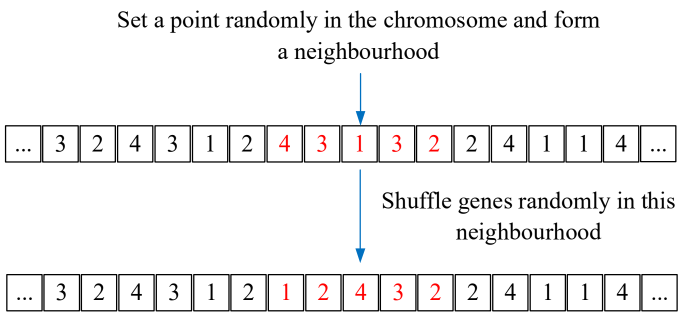

Set a point randomly in the chromosome and form a neighbourhood
... 3 2 4 3 1 2 4 3 1 3 2 2 4 1 1 4 ...
Shuffle genes randomly in this neighbourhood
... 3 2 4 3 1 2 1 2 4 3 2 2 4 1 1 4 ...

Figure 7. The mutation operator in Steps2 and 3.

3.6. Genetic recombination strategies with dynamic gene banks

To further enhance the performance of the algorithm, we propose a genetic recombination strategy with dynamic gene banks for post-conventional genetic operations. Relying solely on the completion time for individual evaluation is narrow in perspective. Therefore, we introduce the concept of "crowding degree". The crowding degree assesses the overall scheduling compactness by calculating the proportion of working time to a machine's completion time. Higher crowding values indicate a more compact process arrangement on the machine. We present a comprehensive evaluation function that holistically assesses individuals by amalgamating crowding degree and final completion time. The formulation of this comprehensive evaluation function implies an inverse correlation between an individual's quality and the evaluation function value. Thus, in each population generation, the individual with the highest composite evaluation function value is generally recognized as a high-quality candidate and assigned as the genetic recombination parent. Conversely, the individual with the lowest composite evaluation function value is typically considered less proficient and designated as the genetic recombination offspring. Nonetheless, a drawback arises wherein individuals possessing non-minimum comprehensive evaluation function values might still carry valuable genes. This may inadvertently overlook quality genes present in other individuals. To address this, we introduce the concept of the "exploration rate". During parent (α) selection, there is a probability $\varepsilon_1 = 70\%$ that will select the individual with the smallest value of the comprehensive evaluation function, a probability $\varepsilon_2 = 10\%$ that will select the second smallest individual, a probability $\varepsilon_3 = 10\%$ that will select the third smallest individuals, and a probability $\varepsilon_4 = 10\%$ that select the fourth smallest individual. The relevant parameters of this strategy are shown in Table 2.

Table 2. Parameters related to the strategy.

Symbol	Description	Count
C_{ik}	Completion time per machine in an individual	—
C_i	Individual completion times	$C_{\max} = \max_{i=1}^{m}\{C_i\}$
W_{ik}	Working time per machine in an individual	—
F_{ik}	Idle time per machine in an individual	—
δ_{ik}	Crowding per machine in an individual	$\delta_{ik} = \frac{W_{ik}}{C_{ik}}$
δ_i	Crowding of individuals	$\delta_i = \frac{\sum_{k=1}^{m}\delta_{ik}}{m}$
φ	Comprehensive evaluation function	$\varphi = \frac{C_i}{\delta_i}$
ε	Exploration rate	—
α	Genetic recombinant parent	—
β	Genetic recombinant daughter	—
λ	Gene segment	—

The comprehensive evaluation function is utilized to identify high-quality individuals and low-quality individuals in the generation. The high-quality individuals are selected as parents, while the low-quality individuals are regarded as daughters. A gene is chosen from the parent and transplanted into the daughter through gene recombination. The fitness of the individual receiving the new gene is evaluated. If the new individual demonstrates an improved fitness value compared to the one prior to recombination, it is retained, and the gene is stored in the quality gene bank. However, if the new individual's fitness value is not better, it is reverted. In subsequent iterations, if there are other high-quality gene segments in the gene bank, they are also removed and recombined into the daughter. The fitness of the individual receiving the new gene is evaluated. If the new individual exhibits superior fitness compared to the one before the recombination, it is retained. Conversely, if the new individual's fitness value is not better, it is restored, and the gene is removed from the gene bank. This process ensures that the most promising genes are utilized in the population and promotes the overall improvement of the GA.

The Pseudo-code of genetic recombination strategies with dynamic gene banks is as follows:

```
Begin
input Chromosome.
Count $C_{\max}$; Record $W_{ik}, F_{ik}$; Count $\delta_{ik}, \varphi$.
Select $\alpha, \beta$ with $\varepsilon$.
Extract gene fragments: select a gene fragment $\lambda$ randomly in $\alpha$ as the High-quality gene
fragment.
        //e.g.  $\alpha = 3\ 2\ 4\ 3\ 1\ 2\ \mathbf{4\ 3\ 1\ 3}\ 2\ 2\ 4\ 1\ 1\ 4$
                $\lambda = \mathbf{4\ 3\ 1\ 3}$
Gene recombination: search in $\beta$, find genes that duplicate the high-quality gene fragment, and
then replace them.
        //e.g.  before replace $\beta = 2\ 3\ 4\ 2\ 1\ 3\ 2\ \mathbf{4\ 1\ 3\ 3}\ 1\ 4\ 1\ 2\ 4$
                after replace $\beta' = 2\ 3\ 4\ 2\ 1\ 3\ 2\ \mathbf{4\ 3\ 1\ 3}\ 1\ 4\ 1\ 2\ 4$
Count $C_{\max}'$ of $\beta'$.
if ($C_{\max}' \le C_{\max}$)
    {Let $\beta'$ replace $\beta$, and put $\lambda$ in the gene bank}
else
    {$\beta$ remain unchanged;
        if (the gene fragment is from the gene bank)
            {it will be removed}
    }
if (any other high-quality genes fragments int the gene bank)
    {repeat the gene recombination mentioned above}
End.
```

3.7. Description of IGA

Addressing the limitations of the GA, we propose the Improved GA. Our approach encompasses several key enhancements to the conventional GA methodology. Firstly, we employ a workpiece-based coding scheme, whereby chromosomes are generated randomly. Subsequently, decoding is conducted to identify idle time within each machine. By harnessing this idle time, certain tasks can be expedited. Furthermore, we introduce the championship method to selectively retain valuable chromosomes. Augmenting this selection process is the integration of an elite retention mechanism. This ensures the preservation of superior candidates during successive iterations. To drive evolution, chromosomes within a generation undergo transformations facilitated by crossover and mutation operators. Moreover, we introduce genetic recombination strategies featuring dynamic gene banks, further promoting generation refinement. The program wouldn't be terminated until the termination condition is triggered. The flowchart for IGA is presented in Figure 8.

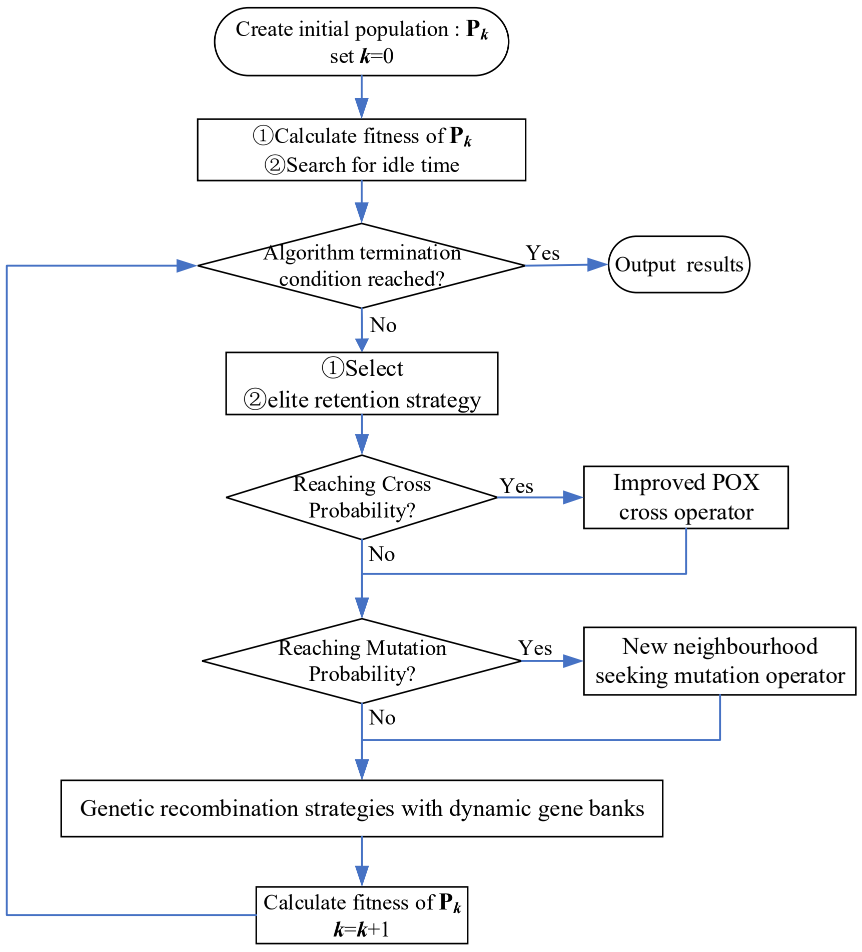

Create initial population : $\mathbf{P}_k$ set k=0
①Calculate fitness of $\mathbf{P}_k$ ②Search for idle time
Algorithm termination condition reached?
Yes
Output results
No
①Select ②elite retention strategy
Reaching Cross Probability?
Yes
Improved POX cross operator
No
Reaching Mutation Probability?
Yes
New neighbourhood seeking mutation operator
No
Genetic recombination strategies with dynamic gene banks
Calculate fitness of $\mathbf{P}_k$ k=k+1

Figure 8. Flowchart for IGA.

4. Algorithm experiment

The improved GA in this paper is programmed in C++ language and compiled and run by using Microsoft Visual Studio 2022 software on a computer with a Windows 11 operating system (Intel Core i7-12700H CPU with the benchmark speed of 2.3GHz and 16G of RAM).

4.1. Testing of crossover and mutation operator

To assess the performance of different crossover operators, we conduct benchmark tests for 20 times, the case information used in the experiments is LA01 which instance of the JSP standard benchmark library. The benchmark tests are performed by using the same program, with the exception of using different crossover operators. The rest of the code remained identical across all tests. Additionally, random number seeds were implemented in the program to ensure randomness and fairness in each test.

According to the experiments, it can be observed that the single-point and multi-point crossover operators met yield nearly identical results in the GA, and neither of them can converge to the optimal solution in each test. Additionally, we simulated the crossover method mentioned in another document, and although it showed promising results, it still falls short compared to the improved crossover method proposed in this paper. In most cases, our proposed method allows the solutions to converge earlier. To illustrate this, we have selected the most representative data and plotted a line chart, as shown in Figure 9.

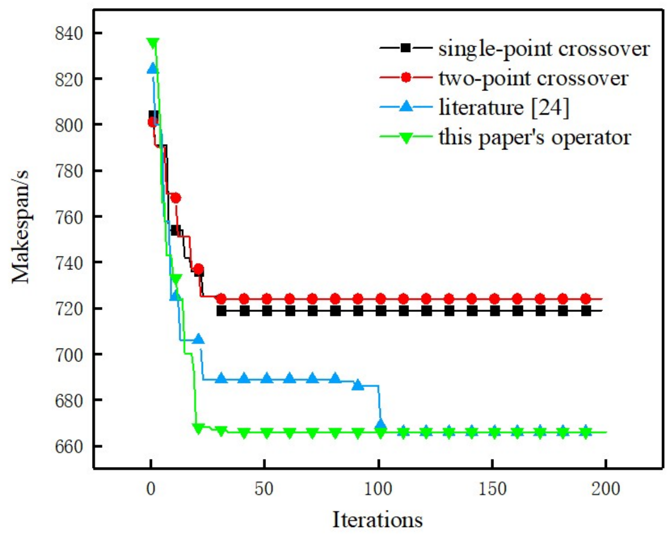


Figure 9. The comparison of convergence curves for different crossover operators.

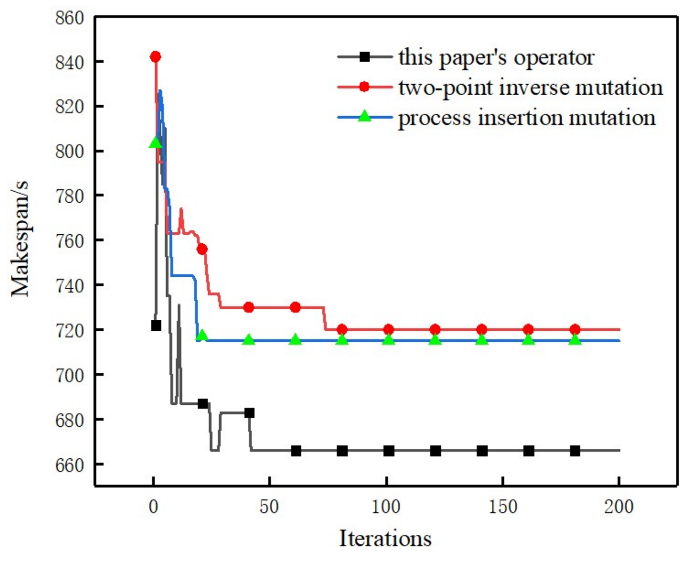


Figure 10. The comparison of convergence curves for different mutation operators.

Furthermore, we also conduct tests on different mutation operators, similarly performing 20 random tests. From the test results, it seems that the performance of the two-point inverse and the process insertion mutation operators are comparable, and both of them fail to guide the algorithm to converge to the optimal solution. In contrast, the mutation operator designed in this paper demonstrates significant performance advantages, in over 70% of the tests, it consistently guides the algorithm to find the optimal solution, highlighting its superior performance. Also, we have selected the most representative data and plotted a line chart, as shown in Figure 10.

4.2. Testing of IGA in basic solution performance

We will first conduct basic performance testing of IGA. The algorithm used in this paper is an improved version based on GA; therefore, we will include the basic GA in the comparison list. Additionally, we have selected the PSO, currently recognized as a classical swarm optimization algorithm, to complement the comparison. In this regard, we select some examples of different sizes from different types of problems from the JSP standard example library, considering the randomness of the algorithm we solve 20 times independently. The termination conditions for the algorithm are as follows: the algorithm finds the known best makespan, or the algorithm reaches one hundred thousand iterations.

The algorithm parameters have been configured as follows: a population size of 100 and a termination condition set either to a predefined threshold of 100,000 iterations or upon reaching a preset optimal solution. To determine the optimal combination of crossover and mutation parameters, a systematic exploration of 9 combinations was conducted. These combinations encompassed crossover probabilities of 0.7, 0.75, and 0.8, mutation probabilities of 0.1, 0.15, and 0.2, along with mutation repetitions ranging from 3, 4, to 5 times. Through a series of comprehensive experiments, it was ascertained that the most favorable outcomes for the algorithm were achieved with a crossover probability of 0.75, a mutation probability of 0.15, and a mutation repetition of 4 times. These specific parameter values yielded the optimal performance for the algorithm.

We will record and compare the following performance metrics of the algorithms. *LB* represents the currently known optimal makespan of the example; t_b represents the best makespan of the 20 tests; t_{av} represents the average makespan of the 20 tests; T_b represents the running time of the program to achieve the best makespan, unit in seconds; I_b represents the iterations of the program to achieve the best makespan. The test results are shown in Table 3.

Table 3. Comparison of the algorithm in this paper with other classical algorithms.

Size	Example	*LB*	IGA			GA		PSO	
			t_b	$T_b(s)$	I_b	t_b	I_b	t_b	I_b
6 × 6	FT06	55	55	0.26	26	59	–	55	34165
10 × 10	FT10	930	930	12.49	4906	1105	–	985	–
20 × 5	FT20	1165	1165	4.02	364	1170	–	1208	–
10 × 5	LA01	666	666	0.26	19	666	78651	676	–
15 × 5	LA06	926	926	0.73	98	928	–	926	48923
20 × 5	LA11	1222	1222	3.96	963	1398	–	1322	–
10 × 10	LA16	945	945	32.78	3419	1057	–	998	–
15 × 10	LA21	1046	1046	90.32	4703	1276	–	1152	–
30 × 10	LA31	1784	1784	3.50	8763	1998	–	1846	–

According to the test results from the benchmark cases, IGA consistently finds the optimal solution for each test case. The program’s running time is relatively short, and the number of iterations is also low. On the other hand, GA and PSO struggle to find the optimal solution within the specified number of iterations, and their final results deviate significantly from the optimal solution. In a few benchmark cases where GA and PSO manage to find the optimal solution, their algorithm requires an exceptionally large number of iterations. As a result, we can conclude that IGA's algorithm performance is superior to that of GA and PSO.

4.3. Testing of IGA in complex solution performance

In order to understand IGA's efficiency in solving large and complex cases, we conducted tests using the benchmark case library containing ABZ instances and ORB instances. Additionally, we used GA and PSO algorithms as a comparison in these tests. By doing so, we aimed to evaluate and compare the performance of IGA, GA, and PSO in tackling these challenging and intricate problem sets. The results of these tests will provide valuable insights into the effectiveness and efficiency of each algorithm in handling large-scale and complex scenarios.

In this test, we have set the termination conditions for each algorithm to be either a maximum runtime of 100 seconds or 200 seconds. By comparing the quality of the solutions obtained by each algorithm within the same running time, we can assess their respective efficiency. Also, considering the randomness of the algorithm we solve 20 times independently, t_{100} represents the algorithm's best makespan obtained when the program runs for 100 seconds; t_{200} represents the algorithm's best makespan obtained when it runs for 200 seconds. The test results are shown in Table 4.

Table 4. Comparison of the algorithm in this paper with other classical algorithms.

Size	Example	*LB*	IGA		GA		PSO	
			t_{100}	t_{200}	t_{100}	t_{200}	t_{100}	t_{200}
10 × 10	ABZ5	1234	1272	1250	1536	1498	1623	1564
10 × 10	ABZ6	943	956	943	1233	1202	1369	1295
20 × 15	ABZ7	656	698	670	992	945	1065	1002
20 × 15	ABZ8	648	696	682	886	842	858	801
20 × 15	ABZ9	678	731	695	798	786	846	796
10 × 10	ORB01	1059	1127	1103	1278	1245	1392	1364
10 × 10	ORB02	888	936	902	1005	986	1132	1096
10 × 10	ORB03	1005	1095	1072	1247	1222	1195	1184
10 × 10	ORB04	1005	1045	1032	1446	1422	1396	1378
10 × 10	ORB05	887	945	906	1020	996	1203	1167
10 × 10	ORB06	1010	1076	1052	1546	1522	1346	1297
10 × 10	ORB07	397	429	408	688	665	732	708
10 × 10	ORB08	899	981	932	1342	1311	1245	1237
10 × 10	ORB09	934	960	942	1462	1432	1365	1324
10 × 10	ORB10	944	986	946	1213	1202	1354	1326

Based on the experimental results, it can be observed that regardless of the runtime being 100 seconds or 200 seconds, IGA consistently outperforms the other two algorithms. Consequently, it can be concluded that in large-scale and complex case tests, the algorithmic efficiency of IGA is significantly higher than that of the other two algorithms.

Table 5. Performance comparison between this algorithm and other algorithms

Size	Example	*LB*	IGA			Document [25]	Document [26]	Document [27]	Document [28]
			t_b	t_{av}	$T_{av}(s)$	t_b	t_b	t_b	t_b
6 × 6	FT06	55	55	55	0.33	55	55	55	55
10 × 10	FT10	930	930	930	50.23	937	930	997	930
20 × 5	FT20	1165	1165	1165	15.31	1180	1165	1196	1165
10 × 5	LA01	666	666	666	0.56	666	666	666	666
10 × 5	LA02	655	655	655	0.72	655	655	655	655
10 × 5	LA03	597	597	597	0.62	597	597	617	597
10 × 5	LA04	590	590	590	1.67	590	590	607	590
10 × 5	LA05	593	593	593	3.56	593	593	593	593
15 × 5	LA06	926	926	926	0.98	926	926	926	926
15 × 5	LA07	890	890	890	2.01	890	890	890	890
15 × 5	LA08	863	863	863	1.37	863	863	863	863
15 × 5	LA09	951	951	951	1.95	951	951	951	951
15 × 5	LA10	958	958	958	2.46	958	958	958	958
20 × 5	LA11	1222	1222	1222	5.46	1222	1222	1222	1222
20 × 5	LA12	1039	1039	1039	8.02	1039	1039	1039	1039
20 × 5	LA13	1150	1150	1150	4.62	1150	1150	1150	1150
20 × 5	LA14	1292	1292	1292	2.18	1292	1292	1292	1292
20 × 5	LA15	1207	1207	1207	3.45	1207	1207	1207	1207
10 × 10	LA16	945	945	945	46.53	956	945	994	945
10 × 10	LA17	784	784	784	16.30	784	784	793	784
10 × 10	LA18	848	848	848	15.26	849	848	860	848
10 × 10	LA20	902	902	904.7	305.62	902	902	912	902
15 × 10	LA21	1046	1046	1046	162.34	1056	1046	1146	1046
15 × 10	LA23	1032	1032	1032	286.24	1036	1032	1033	1032
30 × 10	LA31	1784	1784	1784	8.36	1784	1784	1844	1784
30 × 10	LA32	1850	1850	1850	7.35	1850	1850	1907	1850
30 × 10	LA33	1719	1719	1719	12.09	1719	1719	–	1719
30 × 10	LA34	1721	1721	1721	16.45	1723	1721	–	1721
30 × 10	LA35	1888	1888	1888	12.34	1888	1888	–	1888

4.4. Testing and comparison of IGA and other improved algorithms

In order to compare the solution performance of different algorithms, the test results of the algorithm in this paper, and the test results of other related algorithms in other document were statistically analyzed and compared, using document selected from high-level domestic and international journal document in recent years. The document [25] is a stochastic complex evolutionary algorithm, the document [26] is an improved artificial bee colony algorithm, the document [27] is an agent-based parallel approach for the problem, and the document [28] is a hybrid algorithm of GA and tabu search. We have added some additional performance indicators: t_{av} represents the average makespan of the 20 tests; T_{av} represents the average running time of the program of the 20 tests. The test results are shown in Table 5.

The statistical analysis was carried out in terms of the number of superior and inferior solutions, and the results are shown in Table 6. Compared with the algorithm in [25], 22 cases achieved equal solutions and 7 cases outperformed the algorithm in [25]; compared with [27], 12 cases outperformed the algorithm in [27]; compared with both algorithms in [26,28], the performance is comparable. The results verify the superiority of the proposed algorithm.

Figure 11 shows the Gantt chart for the LA35 operator example derived in this paper.

Table 6. Number of solutions for which the algorithm in this paper outperforms other algorithms.

	Document [25]	Document [26]	Document [27]	Document [28]
Number of equivalent solutions	22	29	14	29
Better than number of solutions	7	0	12	0

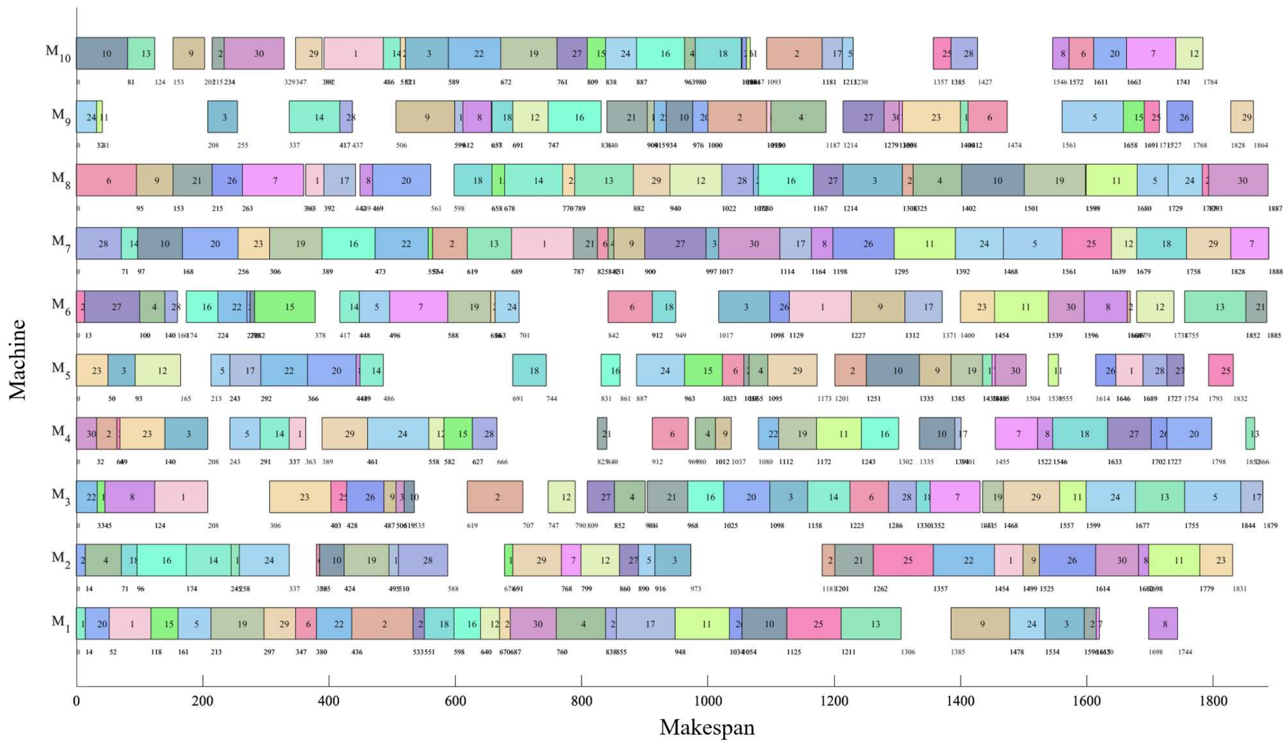


Figure 11. The Gantt chart for the LA35.

5. Conclusions

Many existing basic improved algorithms for JSP often suffer from poor search capability and tend to get trapped in local optimal solutions. In response to this challenge, we propose an enhanced GA that incorporates four key improvements: decoding with idle time searching, an improved POX crossover operator, a novel neighbourhood seeking mutation operator, and a genetic recombination strategy with dynamic gene banks. To assess the effectiveness of IGA, we conduct tests on representative benchmark problems and compare its performance with other algorithms selected from prominent domestic and international journal documents. The computational results demonstrate that our proposed IGA is a highly effective method for tackling JSPs, thus enriching the fundamental theoretical study of JSP.

As the manufacturing industry continues to evolve and confront increasingly complex production environments, our future research will focus on studying JSPs with multiple objectives and constraints. Additionally, we will actively explore better search strategies and develop new hybrid algorithm schemes to further enhance the performance of our algorithms.

Use of AI tools declaration

The authors declare they have not used Artificial Intelligence (AI) tools in the creation of this article.

Acknowledgments

This work was supported by the Research Initiation Foundation of Anhui Polytechnic University (2022YQQ002), Anhui University Research Project (2023AH050935), Anhui Polytechnic University Research Project (Xjky2022002), the Open Research Fund of AnHui Key Laboratory of Detection Technology and Energy Saving Devices (JCKJ2022B01), Key Natural Science Research Projects of Colleges and Universities in Anhui Province (2022AH050978), Anhui Province University Excellent Top Talent Training Project(gxbjZD2022023), Wuhu science and technology project (2022jc26), the Open Research Fund of Anhui Province Key Laboratory of Detection Technology and Energy Saving Devices, Anhui Polytechnic University (JCKJ2021A06) and Anhui Polytechnic University-Jiujiang District Industrial Collaborative Innovation Special Fund Project (2022cyxtb6).

Conflict of interest

The authors declare there is no conflict of interest.

References

1. X. Y. Li, L. Gao, An effective hybrid genetic algorithm and tabu search for flexible job shop scheduling problem, *Int. J. Prod. Econ.*, **174** (2016), 93–110. https://doi.org/10.1016/j.ijpe.2016.01.016

2. X. Y. Li, L. Gao, Q. K. Pan, L. Wan, K. M. Chao, An effective hybrid genetic algorithm and variable neighborhood search for integrated process planning and scheduling in a packaging machine workshop, *IEEE Trans. Syst. Man. Cybern. Syst.*, **49** (2018), 1933–1945. https://10.1109/TSMC.2018.2881686
3. G. H. Zhang, L. Gao, Y. Shi, An effective genetic algorithm for the flexible job-shop scheduling problem, *Expert Syst. Appl.*, **38** (2011), 3563–3573. https://doi.org/10.1016/j.eswa.2010.08.145
4. R. Mellado Silva, C. Cubillos, D. C. Paniagua, A constructive heuristic for solving the Job-Shop Scheduling Problem, *IEEE Latin Am. Trans.*, **14** (2016), 2758–2763. https://10.1109/TLA.2016.7555250
5. G. Vilcot, J. C. Billaut, A tabu search algorithm for solving a multicriteria flexible job shop scheduling problem, *Int. J. Prod. Res.*, **49** (2011), 6963–6980. https://doi.org/10.3182/20060517-3-FR-2903.00038
6. G. H. Zhang, X. Y. Shao, P. G. Li, L. Gao, An effective hybrid particle swarm optimization algorithm for multi-objective flexible job-shop scheduling problem, *Comput. Ind. Eng.*, **56** (2009), 1309–1318. https://doi.org/10.1016/j.cie.2008.07.021
7. J. Zhang, J. Jie, W. L. Wang, X. Xu, A hybrid particle swarm optimisation for multi-objective flexible job-shop scheduling problem with dual-resources constrained, *Int. J. Comput. Sci. Math.*, **8** (2018), 526. https://doi.org/10.1504/IJCSM.2017.088956
8. L. N. Xing, Y. W. Chen, P. Wang, Q. S. Zhao, J. Xiong, A knowledge-based ant colony optimization for flexible job shop scheduling problems, *Appl. Soft Comput.*, **10** (2010), 888–896. https://doi.org/10.1016/j.asoc.2009.10.006
9. J. Wu, G. D. Wu, J. J. Wang, Flexible job-shop scheduling problem based on hybrid ACO algorithm, *Int. J. Simul. Model.*, **16** (2017), 497–505. https://10.2507/IJSIMM16(3)CO11
10. J. Holland, *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*, MIT Press, 1992.
11. C. D. Liou, Y. C. Hsieh, Y. Y. Chen, A new encoding scheme-based hybrid algorithm for minimising two-machine flow-shop group scheduling problem, *Int. J. Syst. Sci.*, **44** (2013), 77–93. https://doi.org/10.1080/00207721.2011.581396
12. J. C. Tang, G. J. Zhang, B. B. Lin, B. X. Zhang, A hybrid algorithm for flexible job-shop scheduling problem with setup times, *Int. J. Prod. Manage. Eng.*, **5** (2017), 23–30. https://doi.org/10.1016/j.proeng.2011.08.689
13. A. Turkyilmaz, S. Bulkan, A hybrid algorithm for total tardiness minimisation in flexible job shop: genetic algorithm with parallel VNS execution, *Int. J. Prod. Res.*, **53** (2015), 1832–1848. https://doi.org/10.1080/00207543.2014.962113
14. I. Ono, M. Yamamura, S. Kobayashi, A genetic algorithm for job-shop scheduling problems using job-based order crossover, in *Proceedings of IEEE International Conference on Evolutionary Computation*, (1996), 547–552.
15. Y. Victor, B. Larisa, T. Andrei, Hybrid flowshop with unrelated machines, sequence-dependent setup time, availability constraints and limited buffers, *Comput. Ind. Eng.*, **56** (2019), 1452–1463. https://doi.org/10.1016/j.cie.2008.09.004
16. H. C. Chang, T. K. Liu, Optimisation of distributed manufacturing flexible job shop scheduling by using hybrid genetic algorithms, *J. Intell. Manuf.*, **28** (2017), 1973–1986. https://doi.org/10.1007/s10845-015-1084-y

17. H. Mokhtari, A. Hasani, An energy-efficient multi-objective optimization for flexible job-shop scheduling problem, *Comput. Chem. Eng.*, **104** (2017), 339–352. https://doi.org/10.1016/j.compchemeng.2017.05.004
18. J. F. Goncalves, M. G. C. Resende, An extended Akers graphical method with a biased random-key genetic algorithm for job-shop scheduling, *Int. Trans. Oper. Res.*, **27** (2014), 215–246. https://doi.org/10.1111/itor.12044
19. C. Y. Zhang, P. G. Li, Z. L. Guan, Y. Q. Rao, A tabu search algorithm with a new neighborhood structure for the job shop scheduling problem, *Comput. Oper. Res.*, **34** (2007), 3229–3242. https://doi.org/10.1016/j.cor.2005.12.002
20. W. Chen, H. Yang, Y. Hao, Scheduling of dynamic multi-objective flexible enterprise job-shop problem based on hybrid QPSO, *IEEE Access*, **7** (2019), 127090–127097. https://10.1109/ACCESS.2019.2938773
21. Y. C. Wang, J. M. Usher, Application of reinforcement learning for agent-based production scheduling, *Eng. Appl. Artif. Intell.*, **18** (2005), 73–82. https://doi.org/10.1016/j.engappai.2004.08.018
22. P. M. Pardalos, O. V. Shylo, An algorithm for the job shop scheduling problem based on global equilibrium search techniques, *Comput. Manag. Sci.*, **3** (2006), 331–348. https://doi.org/10.1007/s10287-006-0023-y
23. M. M. Nasiri, F. Kianfar, A hybrid scatter search for the partial job shop scheduling problem, *Int. J. Adv. Manuf. Technol.*, **52** (2011), 1031–1038. https://doi.org/10.1007/s00170-010-2792-2
24. G. L. Gong, Q. W. Deng, R. Chiong, X. Gong, H. Huang, An effective memetic algorithm for multi-objective job-shop scheduling, *Knowl. Based Syst.*, **182** (2019), 104840. https://doi.org/10.1016/j.knosys.2019.07.011
25. F. Q. Zhao, J. L. Zhang, C. Zhang, J. Wang, An improved shuffled complex evolution algorithm with sequence mapping mechanism for job shop scheduling problems, *Expert Syst. Appl.*, **42** (2015), 3953–3966. https://doi.org/10.1016/j.eswa.2015.01.007
26. N. Sharma, H. Sharma, A. Sharma, Beer froth artificial bee colony algorithm for job-shop scheduling problem, *Appl. Soft Comput.*, **68** (2018), 507–524. https://doi.org/10.1016/j.asoc.2018.04.001
27. A. Leila, Z. Kamran, An agent-based parallel approach for the job shop scheduling problem with genetic algorithms, *Math. Comput. Modell.*, **52** (2010), 1957–1965. https://doi.org/10.1016/j.mcm.2010.04.019
28. J. Xie, X. Y. Li, L. Gao, L. Gui, A hybrid algorithm with a new neighborhood structure for job shop scheduling problems, *Comput. Ind. Eng.*, **169** (2022), 108205. https://doi.org/10.1016/j.cie.2022.108205