Hindawi
Mathematical Problems in Engineering
Volume 2021, Article ID 9951995, 21 pages
https://doi.org/10.1155/2021/9951995

Research Article

Hybrid Algorithm Based on Genetic Simulated Annealing Algorithm for Complex Multiproduct Scheduling Problem with Zero-Wait Constraint

Zhongyuan Liang,[1] **Mei Liu**,[2] **Peisi Zhong**,[1] **Chao Zhang**,[1] **and Xiao Wang**[1]

[1]*Advanced Manufacturing Technology Centre, Shandong University of Science and Technology, Qingdao 266590, China*
[2]*College of Mechanical and Electronic Engineering, Shandong University of Science and Technology, Qingdao 266590, China*

Correspondence should be addressed to Peisi Zhong; pszhong_sdust@163.com

Received 12 March 2021; Revised 26 May 2021; Accepted 7 June 2021; Published 21 June 2021

Academic Editor: Peng-Yeng Yin

Aiming at the complex multiproduct scheduling problem with 0-wait constraint, a hybrid algorithm based on genetic algorithm (GA) and simulated annealing (SA) algorithm was studied. Based on the results of pruning and grading to the operation tree of complex multiproduct, the design structure matrix (DSM) with precedence constraints was established. Then, an initial population coding method based on DSM was proposed and three strategies to optimize the initial population were proposed to improve the quality of the initial population for the situation of multiple operations in the same grade which need to be processed on the same machine. The specific process flow and the setting method of related parameters for the hybrid algorithm were given out. For the infeasible solution produced in the crossover operation, the repair method was proposed. In the decoding process with makespan as the optimization objective, the chromosome genes were classified and the decoding for complex multiproduct scheduling problem with 0-wait constraint was realized through the analysis of its characteristics. The effectiveness of the proposed algorithm for complex multiproduct scheduling problem with 0-wait constraint is verified by the test of related examples in the existing literature.

1. Introduction

Shop scheduling is the core of production management system and one of the most difficult problems in theoretical research. It is committed to providing a profit-maximizing production planning scheme with a series of production constraints and resource constraints to guide the production work in the workshop. The quality of the scheduling plan directly affects the enterprise whether on-time delivery, equipment utilization, inventory control, and other economic indicators. Therefore, manufacturing enterprises need excellent workshop scheduling technology and methods in order to better manage production and win benefits for enterprises. The traditional workshop scheduling problem is mainly to schedule the products which are independent of each other and do not have the sequential constraint relation in the processing stage. Such as flow shop, job shop, and flexible job shop. Johnson [1] was the first to study the flow-shop scheduling problem for sequential processing of two machines. Researchers from all over the world have conducted in-depth research on this issue, and most of them adopted general optimization methods such as mathematics and operations research. Bansal [2] used a modified branch and bound technique to solve the scheduling of n jobs over m machines when the objective is to minimize the sum of completion times of all jobs on the last machine. Ying [3] developed a new mixed integer linear programming (MILP) model and a two-phase enumeration algorithm to improve the best so far exact methods for solving this problem with the objective of minimizing the makespan. Ozolins [4] proposed a new exact algorithm based on the dynamic programming (DP) to solve the no-

wait job-shop scheduling problem (NWJSP) with a makespan objective.

But it becomes infeasible due to the time consuming as the scale of scheduling problem becomes larger and it was proved to be an NP-hard problem by Bansal [2]. All kinds of intelligent algorithms have been developed and applied to shop scheduling problem, and the great potential of these algorithms has been shown gradually. For various scales' scheduling problems, the metaheuristic algorithm attracted extensive attention from scholars at home and abroad. Wu [5] proposed the hybrid ant colony algorithm based on the 3D disjunctive graph model by combining the elitist ant system, max–min ant system, and the staged parameter control mechanism and optimized the flexible job-shop scheduling problem (FJSP) to minimize the longest completion time. Wei [6] proposed a hybrid genetic simulated annealing (SA) algorithm for flow-shop scheduling problems. Shi [7] proposed a hybrid solving method that combines improved extended shifting bottleneck procedure and genetic algorithm (GA) for the assembly job-shop scheduling problem (JSP). Nagano [8] addressed the m-machine no-wait flow-shop scheduling problem with the objective of minimizing makespan subject to an upper bound on total completion time and proposed an iterated greedy with local search algorithm. Mudjihartono [9] proposed a GA-PSO algorithm, which implements it in both parallel and nonparallel modes, and compared to original GA, the GA-PSO gives a 4.58% better solution on average. Liu [10] proposed a multiobjective optimization model aimed at minimizing carbon footprints of all products and makespan and designed a hybrid fruit fly optimization algorithm to solve the proposed model. Ali [11] proposed a new SA algorithm utilizing block insertion and block exchange operators and established a dominance rule to the no-wait flow-shop scheduling problem on m machines with separate setup times. Zhang [12] integrated three metaheuristics algorithms, Shuffled Frog Leaping Algorithm (SFLA), Intelligent Water Drops (IWD) algorithm, and Path Relinking (PR) algorithm, which were put together to solve JSP. Nouiri [13] proposed a two-stage particle swarm optimization (2S-PSO) to solve the FJSP under machine breakdowns with the lowest makespan obtained and also robust and stable schedules. Wu [14] proposed a similarity-based scheduling algorithm for setup time reduction (SSA4STR) and then an improved nondominated sorting GA II (NSGA-II) to optimize the dual-resource-constrained FJSP when loading and unloading time (DRFJSP-LU) of the fixtures are considered. Huang [15] proposed a new and concise Four-Tuple Scheme (FTS) for modelling a job with operation and processing flexibility and an enhanced GA through a more efficient encoding strategy. Yüksel [16] proposed a novel multiobjective discrete artificial bee colony algorithm (MO-DABC), a traditional multiobjective genetic algorithm (MO-GA), and a variant of multiobjective GA with a local search (MO-GALS) for the biobjective no-wait permutation flow-shop scheduling problem. Wu [17] considered an energy-efficient biobjective no-wait permutation flow-shop scheduling problem in the presence of dynamic speed-scaling technique to minimize makespan and total energy consumption through the proposed AM-VNS works by operating some local search on the solutions in WS and updating the corresponding AS iteratively. Wang [18] proposed a particle swarm optimization (PSO) algorithm with improvements to solve the issue of how to reschedule the randomly arrived new jobs to pursue both performance and stability in a job shop.

However, the researchers mainly focus on traditional shop scheduling, flow-shop scheduling, flexible job-shop scheduling, and shop scheduling with some added environment variables. The no-wait constraint is common in flow-shop scheduling problems. Dong [19] introduced the no-wait two-stage flow-shop scheduling problem with the first stage machine having multitask flexibility motivated by a real-world scenario in hospitals. Xu [20] proposed a metaheuristic, self-adaptive memetic algorithm (SAMA for short) for no-wait flow shop with variable processing times and used the evolutionary operators to adaptively control the balance between intensification and diversification of the algorithm. Ying [21] presented a multistart SA with bidirectional shift timetabling (MSA-BST) algorithm for solving the NWJSP. Deng [22] proposed the population-based iterated greedy (PBIG) algorithm for the sequencing subproblem by adopting favourable features of the iterated greedy algorithm and formulated the no-wait job-shop problem with a total flow time criterion based on time difference and decomposes the problem into timetabling and sequencing subproblems.

Although some research work has been done on NWJSP in theory and practice and some useful research results have been obtained, the research work and results mainly focus on the single mass production, and rarely involve the mixed form of multiple variety scheduling, while the multiproduct scheduling problem mainly solves the production scheduling of small batch products, as well as the cooperation between the parts of the relationship between fabrication and assembly. Shi [7] proposed a hybrid solving method that combines improved extended shifting bottleneck procedure and GA for the assembly JSP. Lei [23] proposed a novel encoding method based on the dynamic operation relationship matrix table and solved the integrated scheduling problem with processing and assembly by GA. And he [24] also used the method mentioned above to solve the no-wait problem proposed by Xie [25–28]. Zhao [29] proposed a novel division coding strategy that ensured all initial feasible solutions were designed based on the rank of virtual components, which reflected the sequence constraints of parts processing and assembly.

We summarized the research of the existing papers in Table 1 (the interpretation of acronyms was shown in Table 2). To sum up, the solution method of JSP is a continuous development process. The general direction is to develop from enumeration method to metaheuristic method with the enlargement of problem scale. The application of metaheuristic algorithm not only is shown in solving the workshop scheduling problem but also shows obvious advantages in the path planning problems such as UAV and robot [31, 32]. The workshop scheduling problem also develops from the flow shop at the beginning to the flexible

Table 1: Classification and summarizing of algorithms solving the product scheduling problem.

Development	Publications	Background	Objectives	Algorithms
Enumeration methods	Johnson [1]	Flow shop	Makespan	MP
	Bansal [2]	Flow shop	Makespan	BB
	Ozolins [4]	No-wait job shop	Makespan	DP
	Xie [25–28]	No-wait integrated scheduling problem	Makespan	MP
Metaheuristic algorithm	Wu [5]	Flexible job shop	Makespan	ACO
	Wei [6]	Flow shop	Makespan	GA-SA
	Mudjihartono [9]	Job shop	Makespan	GA-PSO
	Liu [10]	Flexible job shop	Carbon footprint and makespan	FOA
	Nouiri [13]	Flexible job shop	Makespan	Two-stage PSO
	Wu [14]	Flexible job shop	Makespan and total setup time	NSGAII
	Huang [15]	Flexible job shop	Makespan	GA
	Wang [18]	Dynamic job shop	Makespan	PSO
	Nagano [8]	No-wait flow shop	Makespan	Iterated greedy algorithm
	Ali [11]	No-wait flow shop	Total tardiness	SA
	Yüksel [16]	No-wait permutation flow shop	Total tardiness and energy consumption	ABC and NSGAII
	Wu [17]	No-wait permutation flow shop	Makespan and energy consumption	Multiobjective VNS
	Dong [19]	No-wait two-stage flow shop	Makespan	Linear-time combinatorial algorithm
	Xu [20]	No-wait permutation flow shop	Makespan	Self-adaptive memetic algorithm
	Ying [3, 21, 30]	No-wait job shop	Makespan	Multistart SA
	Deng [22]	No-wait job shop	Total flow time	Iterated greedy algorithm
	Shi [7]	Assembly job shop	Makespan	GA
	Lei [23]	Integrated scheduling problem	Makespan	GA
	Zhao [29]	Integrated scheduling problem	Makespan	GA

Table 2

Acronyms	Full term
MP	Mathematical programming
BB	Branch and bound technique
DP	Dynamic programming
ACO	Ant colony optimization
GA	Genetic algorithm
SA	Simulated annealing
PSO	Particle swarm optimization
FOA	Fruit fly optimization algorithm
NSGAII	Fast nondominated sorting genetic algorithm
ABC	Artificial bee colony
VNS	Variable neighborhood search
EDA	Estimation of distribution algorithm

workshop and the workshop scheduling under the constraint conditions. The maximum completion time is still the first optimization objective; of course, there are carbon emissions, machine load, and another multiobjective optimization. The scheduling problem of single small-batch products, as well as the coordination and cooperation relationship between parts and components with high constraint requirements, is a common scheduling problem in small- and medium-sized enterprises. For this kind of problem, the standard metaheuristic algorithm only provides a solution framework, and its specific implementation scheme needs to be greatly adapted. This paper addressed the problem of minimizing makespan on complex multiproduct scheduling problem with 0-wait constraint. To minimize the makespan, a hybrid algorithm based on GA and SA algorithm was adopted. In order to adapt to the complex multiproduct scheduling problem, a coding method based on design structure matrix (DSM) was proposed, which solved the sequence constraints and 0-wait constraints between products. According to the different operations types represented by chromosome, the decoding scheme was given. Through three initialization strategies, the initial population quality was improved effectively. Finally, two complex multiproduct scheduling problems with 0-wait constraints were solved by using a global search of GA and a local search of SA algorithm.

The remaining organisational structure of this paper is as follows. In Section 2, the problem description for complex multiproduct scheduling problem with 0-wait constraint is given. In Section 3, the implementation of the HGSA is described, including the initial population coding based on DSM and optimization strategies to improve initial population quality, selection, crossover, mutation operator, and decoding algorithm. In Section 4, the specific examples given in the relevant literature are tested and relevant comparative analyses are reported. Finally, the conclusions of this paper and future research work are given.

2. Problem Description

The research on the traditional JSP solves the scheduling problem of pure processing after decomposing the product into unconstrained jobs, which is more suitable for large quantities of the same product. Complex product problem deals with the process scheduling problem of complex products with constraints among the workpiece, which processes the processing and assembly of complex products together. Generally speaking, in order to simplify the complex product scheduling problem, the processing and assembly procedures are unified into operations, and the processing and assembly equipment or stations are unified into machines. Complex multiproduct scheduling problem referred to the simultaneous scheduling of multiple varieties of complex products. Given the product set $P = \{P_1, P_2, \ldots, P_l\}$, the product $P_i (P_i \in P, 1 \le i \le l)$ has operations $\{O_{i_1}, O_{i_2}, \ldots, O_{in}\}$, and all the operations are processed on machine set $M = \{M_1, M_2, \ldots, M_m\}$ with the following constraints:

(1) Each operation O_{ij} can only be processed by one machine M_k at a time and each machine M_k can only process one operation O_{ij} at a time, $1 \le j \le n, 1 \le k \le m$.

(2) Once a machine processes an operation, it cannot process other operations until the operation is completed.

(3) Sequence constraints exist among operations and follow the relationship given in the product operation tree, and any operation cannot be interrupted until it is completed, while the processing time of each operation is known in the product operation tree.

(4) All machines are idle at time 0.

On the basis of the above description, the complex multiproduct scheduling problem is to reasonably arrange the operation processing sequence on each machine with the completion time of all the products being as small as possible. If C_{ij} indicates the completion time of the operation O_{ij}, so the objective function is

$$\min C_{\max} = \min\left(\max_i^l\left(\max_j^n\left(C_{ij}\right)\right)\right), \tag{1}$$

subject to

$$S_{ij} \ge 0, \tag{2}$$

where $1 \le i \le l$ and $1 \le j \le n$. S_{ij} is the start time of operation O_{ij}.

$$S_{ij} + P_{ij} = C_{ij}, \tag{3}$$

where $1 \le i \le l$ and $1 \le j \le n$. P_{ij} is the process time of operation O_{ij} and C_{ij} is the completion time of operation O_{ij}.

$$C_{ij} \le S_{pq}, \tag{4}$$

where $1 \le i \le l$, $1 \le j \le n$, $1 \le p \le l$, $1 \le q \le n$, and O_{ij} is prior to O_{pq}.

$$P_{ijk} + P_{pqk} \le \left|C_{pqk} - S_{ijk}\right|, \tag{5}$$

where $1 \le i \le l$, $1 \le j \le n$, $1 \le p \le l$, $1 \le q \le n$, and $1 \le k \le m$. P_{ijk} is the process time of operation O_{ij} on machine M_k.

Constraints (2)–(4) describe sequential constraints between operations and operations cannot be interrupted. Meanwhile, all time nodes are unique; that is, the same operation can only be processed once by a machine. Constraint (5) is the constraint to machine that each machine can only process one operation at a time.

However, there are some other constraints in the production of complex products, such as zero-wait constraint between operations. Zero-wait constraint between operations means that after a certain operation is finished, it must immediately enter the next operation to process; that is, there should be no time gap between operations. The 0-wait constraint can be described in constraint

$$\sum_{j=1}^{h} P_{ij} = C_{ih} - S_{ij}, \tag{6}$$

where $1 \le i \le l$, $1 \le j \le n$, $1 \le h \le n$, and $O_{ij}, \ldots, O_{ih} \in NW_t$. NW_t is a set of operations with the same 0-wait constraints.

Figure 1 shows the operation tree of products A and B. Each rectangular box represents an operation. The data in the rectangular box represent the product name, operation number, machine number, and processing time in turn, and arrows represent sequential constraints. The dotted box represents the 0-wait constraint, which refers to the completion of the previous operation that must be seamless into the next operation, such as injection molding, steelmaking, and hot assembly.

3. Methods

As a typical iterative algorithm, GA and SA algorithm can be used to solve the JSP. GA is based on the mechanism of biological evolution. It starts from the string set of problem solution and has a strong global searching ability. SA algorithm originates from the principle of solid annealing and is a probabilistic-based algorithm. It starts from the current solution that produces a new solution in the solution space and has a strong local searching ability. If the single-threaded search of SA algorithm is changed into multithreaded search, that is to say, add the population iteration idea of GA, then the hybrid algorithm will have the ability of global search and local search. The hybrid genetic SA algorithm uses a large number of samples as a possible solution to the problem rather than a single sample as a possible solution to the problem, which will greatly increase the convergence rate. At the same time, probabilistic acceptance of new solutions can also solve the population diversity and prevent entering local optimal. To sum up, the hybrid algorithm complements the GA and SA algorithm and has more advantages in solution than the single algorithm. For the above analysis of the characteristics of the complex multiproduct 0-wait constraint problem, in this paper, we adopted the

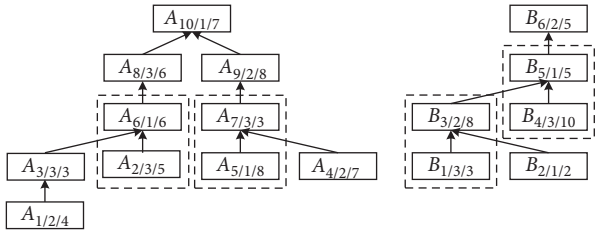


Figure 1: Operation tree of products A and B.

hybrid algorithm based on GA and SA algorithm (HGSA) to solve the problem.

According to the research of the operation tree, first, we simplify each operation into a virtual workpiece satisfying the sequential constraint relationship, and on this basis, we use DSM to generate the initial population. At the same time, we adopt the proposed three optimization strategies and reverse coding to improve the initial population quality and then enter the solution searching process using HGSA. The flowchart of the algorithm is shown in Figure 2.

3.1. Initial Population Coding Based on DSM

3.1.1. Pruning Method and Grading. Directed graph is one of the simplest methods to express the process. It mainly consists of the nodes representing tasks or features and the directed edges indicating the transmission of requirement information between the nodes. The operation tree is also representation of a directed graph. Pruning the product operation tree can simplify the model and improve the coding efficiency. Simply put, pruning is the process of integrating the single line operations or 0-wait operations into a virtual workpiece.

The steps of pruning method are as follows:

Step 1. Calculate the node in-degree in the directed graph corresponding to the product operation tree and mark the nodes with in-degree being 0.

Step 2. Starting from a node with in-degree being 0, find the next node along the directed edge until reaching another node with in-degree being bigger than 1 or another node belonging to 0-wait operations. However, the above method is applicable; the starting node does not belong to the 0-wait operations. On the contrary, if the starting node belongs to 0-wait operations, end the searching process after the last 0-wait operations. Then, get a new operation tree.

Step 3. Repeat Step 2 to the new operation tree until reaching the last operation in the operation tree.

Virtual workpieces are graded based on the completion of pruning, using the method of calculating the task height value in a directed acyclic graph [33]. The grade of a virtual workpiece is defined as

$$\text{grade}(J_i) = \begin{cases} 0, & \{\text{pred}(J_i)\} = \phi, \\ \max\{\text{grade}(\text{pred}(J_i))\} + 1, & \text{otherwise.} \end{cases} \tag{7}$$

$\text{grade}(J_i)$ is the grade of virtual workpiece J_i. $\text{pred}(J_i)$ is the immediate predecessor virtual workpiece of J_i.

The serial number i of virtual workpiece J_i is defined as follow:

Step 1. Calculate the number of virtual workpieces less than each grade.

Step 2. In the same grade, number the virtual workpiece as 1 on the far left, then +1 on the second far left, and so on till reaching virtual workpiece on the far right.

Step 3. In the same grade, add the number obtained from Step 1 and Step 2, and get the serial number i of current virtual workpiece J_i.

Based on the definition above, if the grade number of one virtual workpiece is bigger than another, that means its serial number is bigger.

As shown in Figure 3, the shadow represents the operation with in-degree of 0, that is, the starting operation. The dotted box represents the 0-wait constraint. We can get the mapping relation such as $J_1 = \{A_1, A_3\}$, $J_4 = \{A_2, A_6\}$, and $J_7 = \{A_8\}$.

3.1.2. Coding Based on DSM. The solution of JSP is the sequence of jobs, and the sequence can get the start time and end time of each job on each machine. Based on this, the virtual workpieces can be sorted and solved according to the constraint relationship. Donald Steward [34] introduced structural design matrices (DSM) to analyse information flows in 1981. In this paper, we proposed an initialization population coding method for operations based on DSM, which contains the sequence constraints between virtual workpieces.

Figure 4 shows the specific principle of DSM. Building the DSM, firstly, the virtual workpieces are arranged in ascending order according to the grade from top to bottom and left to right. In principle, there is no sequence requirement for virtual workpieces in the same grade, but they are generally in ascending order according to the serial number of jobs. In DSM, the row number of the element represents the serial number of the virtual workpiece, and the column number of the element represents the serial number of the virtual workpiece corresponding to the immediate predecessor of the current virtual workpiece. For diagonal elements, if it is a virtual workpiece with 0-wait constraint, put 1 in; else, put the number of operations in this virtual workpiece in. For nondiagonal elements, if it is a virtual workpiece with 0-wait constraint, which means that there will be at least 2 operations in the virtual workpiece, find the serial number of its immediate predecessor, and in this column, put the serial number of the operation in current virtual workpiece in. On the contrary, if it is a virtual workpiece without 0 wait constraint, then the pruning rules determine that only the 1st operation in it has immediate predecessor, and put 1 in. For other elements, put 0 in. To sum up, it can be summarized as

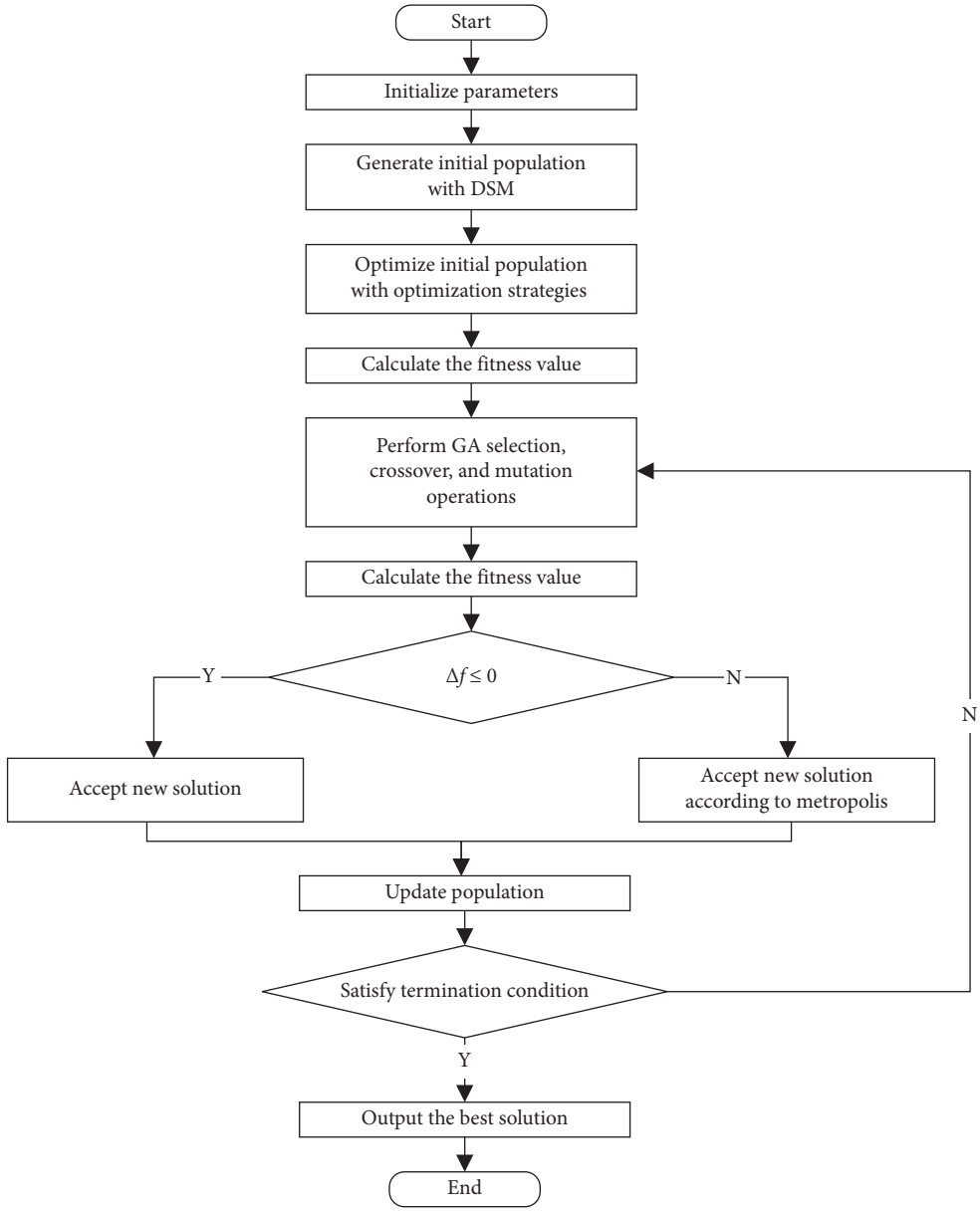


Figure 2: Algorithm flowchart of HGSA.

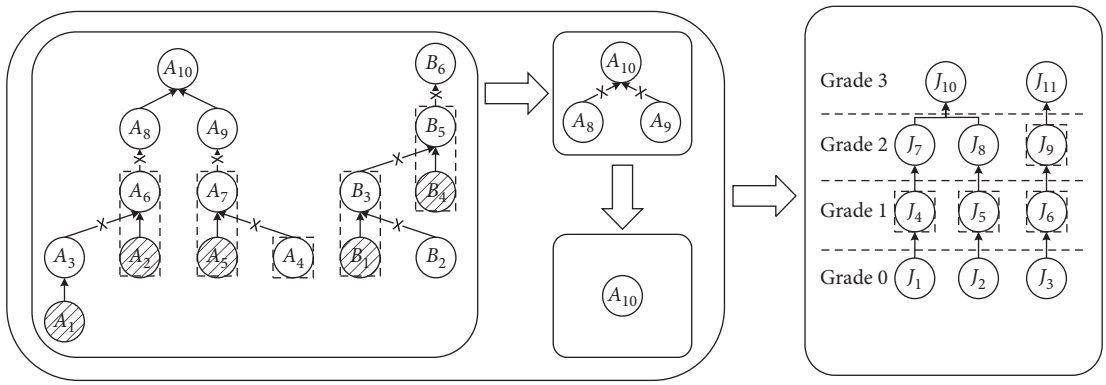


Figure 3: Pruning process and grading result.

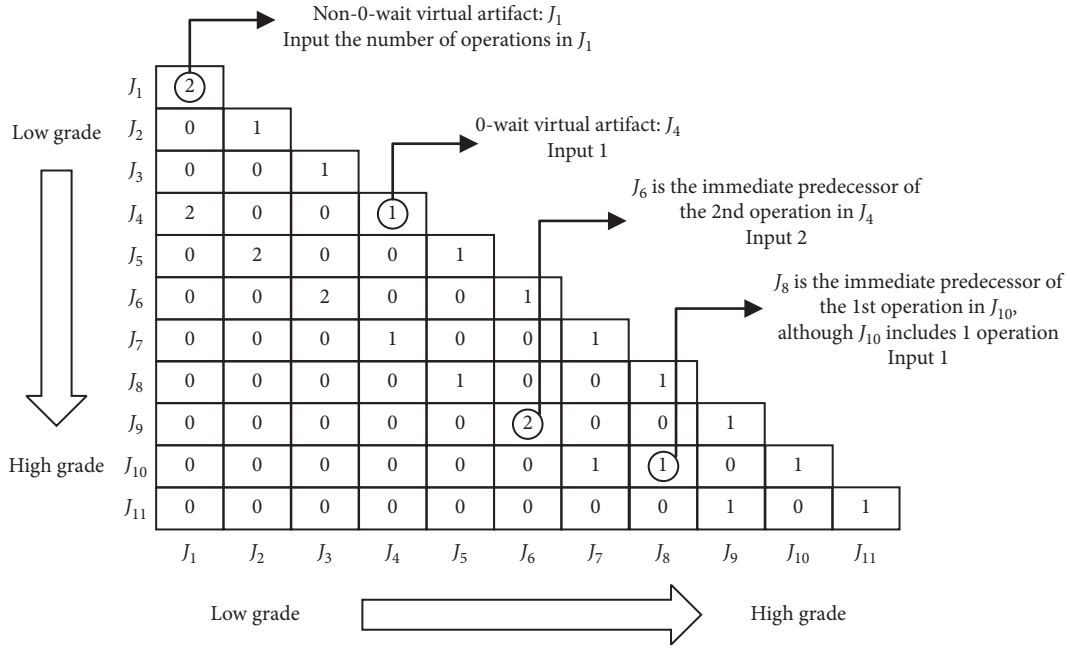


Figure 4: The DSM of virtual workpieces.

$$\mathrm{DSM}(i,i) = \begin{cases} 1, & J_i \text{ is } 0-\text{wait virtual workpiece}, \\ n_{Ji}, & J_i \text{ is non}-0-\text{wait virtual workpiece}, \end{cases} \tag{8}$$

$$\mathrm{DSM}(i,j) = \begin{cases} k, & J_j \text{ is immediate predecessor of } O_k \text{ in } J_i, \\ 0, & \text{otherwise}. \end{cases} \tag{9}$$

Then, get the DSM of the above problem as

$$\mathrm{DSM} = \left(a_{ij}\right)_{n\times n} = \begin{bmatrix} 2&0&0&0&0&0&0&0&0&0&0 \\ 0&1&0&0&0&0&0&0&0&0&0 \\ 0&0&1&0&0&0&0&0&0&0&0 \\ 2&0&0&1&0&0&0&0&0&0&0 \\ 0&2&0&0&1&0&0&0&0&0&0 \\ 0&0&2&0&0&1&0&0&0&0&0 \\ 0&0&0&1&0&0&1&0&0&0&0 \\ 0&0&0&0&1&0&0&1&0&0&0 \\ 0&0&0&0&0&2&0&0&1&0&0 \\ 0&0&0&0&0&0&1&1&0&1&0 \\ 0&0&0&0&0&0&0&0&1&0&1 \end{bmatrix}. \tag{10}$$

The steps of the chromosome encoding method based on DSM are as follows:

Step 1. Get set S of optional jobs. According to the characteristics of DSM, if all elements are 0 in row i except a_{ii}, then put $J_i$into S.

Step 2. Select one job J_k randomly in the set S as the current coding job and put gene k at the latest station of the chromosome. Meanwhile, update $a_{kk} = a_{kk} - 1$. If the new $a_{kk} = 1$, perform Step 3. If the new $a_{kk} = 0$, update 0 to all elements in column k of DSM, and then perform Step 3.

Step 3. Update DSM. Repeat Step 1 until DSM = 0.

Step 4. Insert 0-wait genes into the chromosome. If there are m operations in one 0-wait virtual workpiece, then repeat $m - 1$ times insert gen m after gen m. End Step 4 till all 0-wait jobs complete gene insertion.

After Step 3, the resulting chromosome contains all operations' genes of non-0-wait virtual workpieces and 1 gene of each 0-wait virtual workpiece, for instance, 3-6-2-1-5-1-4-8-7-10-9-11, in which, there are two genes "1" and only one "4," "5," "6," and "9." After Step 4, the final coding chromosome is 3-6-6-2-1-5-5-1-4-4-8-7-10-9-9-11. If a number i appears twice, it means that the virtual workpiece has 2 operations, the 1st number i represents the 1st operation in J_i, and the 2nd number i represents the 2nd operation in J_i.

3.2. Optimization Strategies to Improve Initial Population Quality. The quality of initial population has a great influence on the speed and quality of GA. At present, most of the literature generally adopts the random initialization method, which makes the quality of the initial solution unstable, especially when multiple operations need to be processed on the same machine. To solve this problem, combined with the characteristics of complex multiproduct scheduling problem, we proposed optimization strategies for the initial population. With the population produced by optimization strategies liberated into the initial population, we expect to improve the convergence speed of the algorithm.

When multiple operations in set S of optional jobs need to be processed on the same machine, obviously, these operations are in the same grade, and the optimization strategies are as follows.

For operations O_i and O_k, where O_i and O_k are both belong to leaf node operations, define L_i as the number of the operations from O_i to root node operation. Define T_i as the time of the operation O_i. Define ST_i as the sum time of the operation from O_i to root node operation (Figure 5).

(1) More-task operation first (Figure 5(a)). If $L_i \geq L_k$, then priority process O_i. If the priority is O_k then the number of operations that can be processed in parallel will be less, and the time of parallel processing will be increased.

(2) Long-path operation first (Figure 5(b)). If $L_i = L_k$, $ST_i \geq ST_k$, then priority process O_i. In oriented graph, the path with the maximum path length from the start point to the end point is the critical path. Of cause, the path where O_i located is the critical path, and in the scheduling problem, it determines the value of makespan.

(3) Short-time operation first (Figure 5(c)). If $L_i = L_k$, $ST_i = ST_k$, $ST_i \geq ST_k$, then priority process O_i. As the explanation in (2), either the path where J_i located or the path where O_k located can be selected to be the critical path. Therefore, when the operation with short processing time (O_i) is given priority, the operation with long processing time (O_k) enters the parallel processing state, which is more conducive to shorten the total processing time.

3.3. Select, Cross, and Mutate

3.3.1. Selection Operator. In GA, the commonly used selection operators are rotating selection operator, rotating selection operator with ranking, random consistent selection operator, tournament selection operator, and so on. This paper adopts tournament selection operator. During the selection of the tournament selection operator, k individuals are randomly selected from the population, and the one with the best fitness among the k individuals is identified as the optimal individual. This optimal individual is an individual in the next generation population, and the process repeats n times to produce a new population.

3.3.2. Crossover Operator. The purpose of crossover is to make use of the parent individuals to generate new individuals after a certain combination of operations and to search the solution space efficiently on the basis of minimizing the probability that the effective mode is destroyed. Zhang [35] adopted POX (multiparent precedence operation crossover) to for FJSP. Zhao [29] applied POX and IPOX to GA with virtual component level division coding. However, it is easy to produce infeasible solutions when POX is directly applied to the assembly JSP, so it is necessary to repair the chromosome with infeasible solution.

The steps with repair to infeasible solution based on POX are shown in Figure 6 and are as follows:

Step 1. Randomly select a virtual workpiece as the intersection point. Let us say J_8 is selected.

Step 2. Divide the set of virtual workpieces into two sets according to the node workpiece: one is the set (Set_A) of virtual workpieces in front of the node and the other set (Set_B) is the rest virtual workpieces. Set_A = {2, 5, 8}, and Set_B = {1, 3, 4, 6, 7, 9, 10}.

Step 3. The genes in Set_A of parent chromosome are passed on to offspring in situ and the genes in Set_B of the other parent chromosome are passed on to offspring supplementally.

Step 4. The offspring chromosome with the greater serial number of the selected node in parent chromosome needs to be repaired. The offspring chromosome C2 needs to be repaired because J_{10} is earlier than J_8.

(a) Get set (Set_C) of virtual workpieces which must be processed after the selected node workpiece.

(b) Extract the genes in Set_C of the offspring chromosome and insert them after the gene of selected node workpiece, and the follow-up gene complements the blank gene.

3.3.3. Mutation Operator. In terms of the ability to generate new individuals in the process of GA, crossover operation is the main method to generate new individuals, while mutation operation is the auxiliary method to generate new individuals. Combined with the characteristics of 0-wait complex multiproduct scheduling problem, this paper presents a new insertion mutation operator. The steps are as follows (Figure 7):

Step 1. Randomly select a virtual workpiece as the intersection point. Let us say J_8 is selected.

Step 2. Divide the set of virtual workpieces into two sets according to the node workpiece: one is the set (Set_A) of virtual workpieces in front of the node and the other set (Set_B) is the rest virtual workpieces. Set_A = {2, 5, 8}, and Set_B = {1, 3, 4, 6, 7, 9, 10}.

Step 3. Insert the elements in Set_A into the offspring chromosome in turn according to the original procedure. The principle of Step 3 is shown in Figure 7.

(a) Get the number (Num_A) of workpieces in Set_A, and get the serial number (SN_P) of the tight after operation of the point workpiece in parent chromosome. Count_A = 0.

(b) Select the elements in Set_A in order. Count_A = Count_A + 1. If the current workpiece is the first element in Set_A, then insert it at [1, SN_P − Num_A]. Otherwise, get the serial number (SN_pre) of the immediate predecessor workpiece of it in offspring chromosome and insert it at [SN_pre + 1, SN_P − (Num_A − Count_A) − 1].

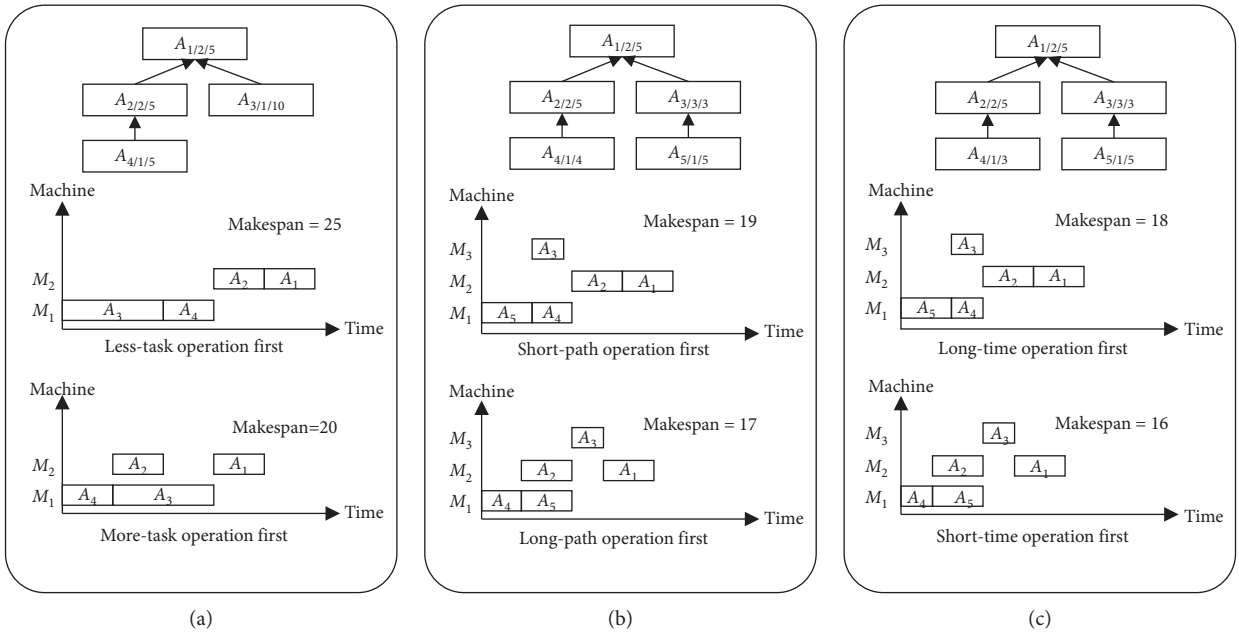


Figure 5: The optimization strategies.

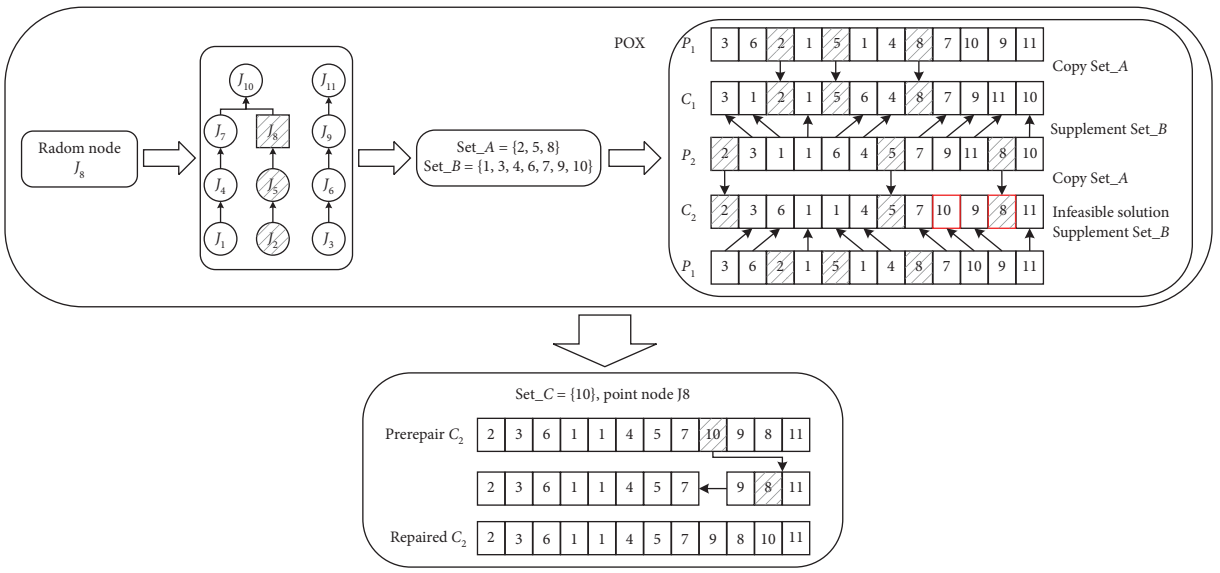


Figure 6: The POX crossover.

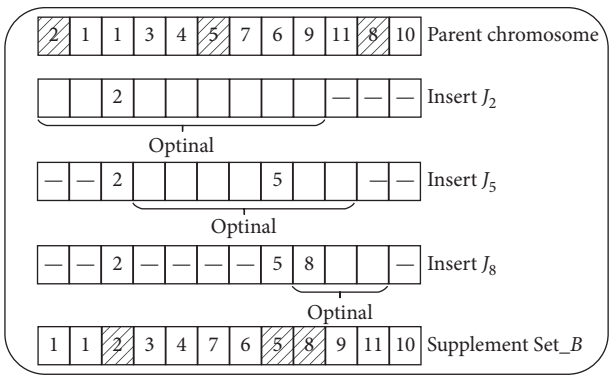


Figure 7: Step 3 of mutation.

(c) Repeat (b) until Count_A = Num_A.

(d) Supplement the elements in Set_B to the blank of offspring chromosome.

3.4. Decoding. Decoding is the process of calculating the start and end time of each operation according to chromosome and determining the maximum completion time (makespan). In this paper, based on the initial operation of leaf node, the genes are divided into two categories, according to the machine counter and workpiece counter to decode.

The notations are defined as follows:

O_i, the current decoding operation represented by the ith gene.

O_i_pre, the immediate predecessor operation of the current decoding operation O_i.

O_leaf, leaf node initial workpiece set; that is, there is no precedence operations, which only refers to the first operation in the virtual workpiece

M_counter, record the number of operations processed by the machine where the current decoding operation is located, including the current operation.

O_counter, record the number of operations in the virtual workpiece where the current decoding operation is located, that is, the serial number in the virtual workpiece.

No_wait, 0-wait operations set. For uniline no-wait constraint, that means all the operations in the same virtual workpiece have only one immediate predecessor operation, and in DSM, that is there is only one "1" element except the diagonal element in a row. For multiline no-wait constraint, that means allowing an operation in the virtual workpiece to have more than one immediate predecessor operation, and in DSM, there are non-0 and non-1 elements in a row.

S_i, start time of the ith gene.

P_i, process time of the ith gene.

E_i, end time of the ith gene.

E_mb, end time of the last operation on the machine where the current decoding process is located.

E_ob, end time of the last operation in the virtual workpiece where the current decoding operation is located.

E_opre, end time of the immediate predecessor operation of the current decoding operation.

The process of decoding, to put it simply, is to determine the operation type of current gene in the chromosome and determine when and which machine can start the operation. The M_counter and O_counter can help us to understand the end time of E_mb, E_ob, and E_opre, and by comparing their values, the start time of current operation can be obtained. It is worth noting that active decoding is used here so that when it is found that there is enough free time in front of the machine and the operations sequence requirements are met, the process can be inserted forward and the gene sequence can be corrected. For the process with 0-wait constraint, the scheduled preoperation is updated according to the situation to make sure that there is no time gap between them. The specific decoding algorithm is in Algorithm 1 and the schematic diagram of decoding is in Figure 8.

After decoding the chromosome, the start time and completion time of each operation per machine can be obtained, and the corresponding scheduling Gantt chart can be drawn. The maximum completion time of the last operation of each machine is the value of the objective function.

3.5. Metropolis in Hybrid Algorithm. The setting of the initial value of temperature T_0 is one of the important factors affecting the global search performance of the SA algorithm. If T_0 is high, it is possible to find the global optimal solution, but it takes a lot of computation time. Otherwise, the computation time can be saved, but the global search performance may be affected. This paper used the following formula to set the initial temperature:

$$T_0 = U_{\max} - U_{\min}, \tag{11}$$

where $U_{\max}$ is the maximum solution in the contemporary population. $U_{\min}$ is the minimum solution in the contemporary population.

Based on the above description, the acceptance probability of the solution in the nonoptimal direction is

$$p = \exp\left(\frac{-\Delta f}{T_0}\right), \qquad \Delta f = C_{\max}(k+1) - C_{\max}(k), \tag{12}$$

where k represents the number of iterations.

4. Experiments Results and Analysis

The existing international benchmark test examples can be directly used for JSP. However, the standard test case library for complex product scheduling problem has not been established, especially the complex multiproduct scheduling problem with 0-wait constraint. In order to verify the performance of the algorithm proposed in this paper, examples found in relevant literature are selected for testing.

The algorithm was coded in MATLAB language. The computer CPU (Intel Core i5-4460) locks at 3.2 GHz and the RAM is 8 GB.

4.1. Case 1

Case 1. is an example of the problem in reference of Xie [25]. Figure 1 is its operation tree of product. Figure 3 is its pruning process and grading result. The DSM representing the sequential constraint relationship between operations is shown in equation (10). For this case, the algorithm parameters were set as follows: the initial population size was

```
Input: Operation chromosome, Process time chromosome, Machine chromosome;
Output: Makespan;
(1)  for (i ← 1; i ≤ length of operation chromosome; i++)
(2)    Case 1 (O_i belongs to O_leaf)
(3)      if (O_counter = 1)
(4)        if (M_counter = 1)//Figure 8-①
(5)          S_i = 0;
(6)        else//Figure 8-②
(7)          S_i = E_mb;
(8)        end if
(9)      else
(10)       if (M_counter = 1)//Figure 8-③
(11)         S_i = E_ob;
(12)       else//Figure 8-④
(13)         S_i = max (E_mb, E_ob)
(14)       end if
(15)     end if
(16)     E_i = S_i + P_i;
(17)   Case 2 (O_i does not belong to O_leaf)
(18)     if (O_counter = 1)
(19)       if (M_counter = 1)
(20)         if (O_i does not has immediate predecessor operation)//Figure 8-⑤-1
(21)       S_i = 0;
(22)       else//Figure 8-⑤-2
(23)         S_i = E_opre;
(24)       end if
(25)     else
(26)       if (O_i does not has immediate predecessor operation)
(27)         if (O_i cannot be inserted forward)//Figure 8-⑥-1, Figure 8-⑥-2
(28)           S_i = E_mb;
(29)         else//Figure 8-⑥-3
(30)           S_i = the starting point of the idle time that can be inserted;
(31)         end if
(32)       else//Figure 8-⑥-4
(33)         S_i = max (E_mb, E_opre);
(34)       end if
(35)   end if
(36) else
(37)       if (O_i does not belong to No-wait)
(38)         if (M_counter = 1)//Figure 8-⑦
(39)         S_i = E_ob (or E_opre);
(40)       else//Figure 8-⑧
(41)         S_i = max (E_mb, E_ob (or E_opre));
(42)           end if
(43)       else
(44)         if (O_i belongs to uniline No-wait)
(45)           if (M_counter = 1)//Figure 8-⑨-1
(46)             S_i = E_ob (or E_opre);
(47)         else//Figure 8-⑨-2
(48)               S_i = max (E_mb, E_ob (or E_opre));
(49)               E_{i-1} = S_i;
(50)               S_{i-1} = E_{i-1} − P_{i-1};
(51)           end if
(52)         else
(53)           if (M_counter = 1)//Figure 8-⑩-1
(54)             S_i = max (E_ob, E_opre);
(55)             if (E_ob < E_opre)
(56)               E_{i-1} = S_i;
(57)               S_{i-1} = E_{i-1} − P_{i-1};
(58)           end if
(59)         else//Figure 8-⑩-2
```

Algorithm 1: Continued.

```
(60)                        S_i = max (E_mb, E_ob, E_opre);
(61)                        if (E_ob < E_opre ||E_ob < E_opre)
(62)                            E_{i-1} = S_i;
(63)                            S_{i-1} = E_{i-1} − P_{i-1};
(64)                        end if
(65)                    end if
(66)                end if
(67)            end if
(68)        end if
(69)        E_i = S_i + P_i;
(70)    end for
(71)    Makespan = max_i^{o-num} (E_i)
```

Algorithm 1: Decoding.

20, the maximum genetic algebra was 50, the crossover probability was 0.9, and the mutation probability was 0.05.

In the process of solving 10 times, the average makespan was 41, and the average solution time was 2.175 s. Figure 9 is the population distribution map of Case 1. In order to better represent the population density in the same generation, individuals were moved to the neighborhoods in the mapping, so that individuals with the same fitness are not concentrated in one point. It shows that the individual diversity of the primary population was relatively large, and there was no most solvable one. With the increase in the number of iterations, individuals gradually moved to the optimal solution. However, in a certain generation of population, the existence of mutation individuals can prevent the population from falling into the local optimal solution.

Figure 10 is the population evolutionary process of Case 1. In the whole evolutionary process, the individuals of the population moved evidently to the optimal solution twice in the 4th and 11th generations, respectively. Finally, the optimal solution was obtained in the 12th generation. With the change of population iteration times, the mean and variance of individual fitness also showed good convergence. The fluctuation of variance can reflect the existence of population richness, so as to avoid falling into the local optimal solution.

Figure 11(a) is the Gantt chart of the optimal scheduling result obtained by this algorithm and the optimal solution result is 41. At the same time, this result was also obtained in reference of Guo [24]. Figure 11(b) is the Gantt chart of the optimal scheduling result obtained by the reference of Xie [28] and the optimal solution result was 45. Figure 11(c) is the Gantt chart of the optimal scheduling result obtained by the reference of Xie [25] and the optimal solution result was 46. Figure 12 is the population evolutionary process of diffident algorithm. Here, the algorithm parameters were all set as follows: the initial population size was 50, the maximum genetic algebra was 50, the crossover probability was 0.9, and the mutation probability was 0.05. As shown in the picture, the algorithm proposed in this paper has high initial population quality and the fastest convergence speed. The reference of Guo [24] has the second fastest convergence rate. The convergence speed of the standard GA is the slowest. To sum up, this is because the algorithm proposed in this paper adds optimized high-quality individuals to the initial population and combines with the greedy mechanism of SA algorithm, which makes this algorithm better than other algorithms.

4.2. Case 2

Case 2. is an example of the problem in reference of Xie [27]. Figure 13 is its operation tree of product. Figure 14 is its pruning and grading result. The DSM representing the sequential constraint relationship between operations is shown in equation (9). For this case, the algorithm parameters were set as follows: the initial population size was 30, the maximum genetic algebra was 50, the crossover probability was 0.9, and the mutation probability was 0.1.

$$\mathrm{DSM} = \left(a_{ij}\right)_{n\times n} = \begin{bmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 2 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 2 & 0 & 0 & 0 & 0 & 0 & 4 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 2 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 2
\end{bmatrix}. \tag{13}$$

In the process of solving 10 times, the average makespan is 27.4 (27 appeared 8 times and 29 appeared twice), and the

Figure 8: Schematic diagram of decoding.

average solution time is 4.636 s. The optimal scheduling result obtained by this algorithm is 27. Figure 15 is 6 different Gantt charts of the optimal scheduling results obtained by the proposed algorithm in this paper. The reference of Guo [24] set the population size to 200. In the process of solving 10 times, the average makespan is 27.5, the

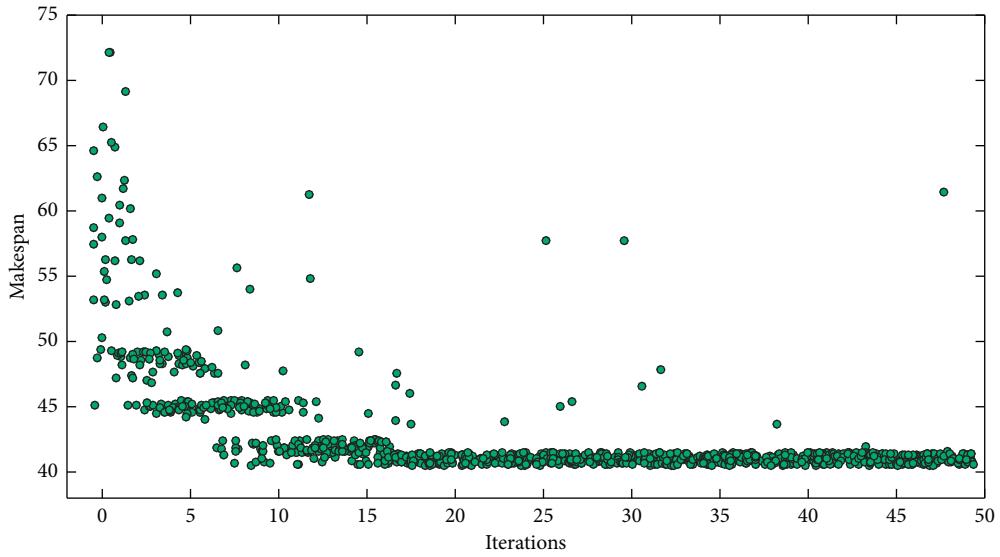


Figure 9: The population distribution map of Case 1.

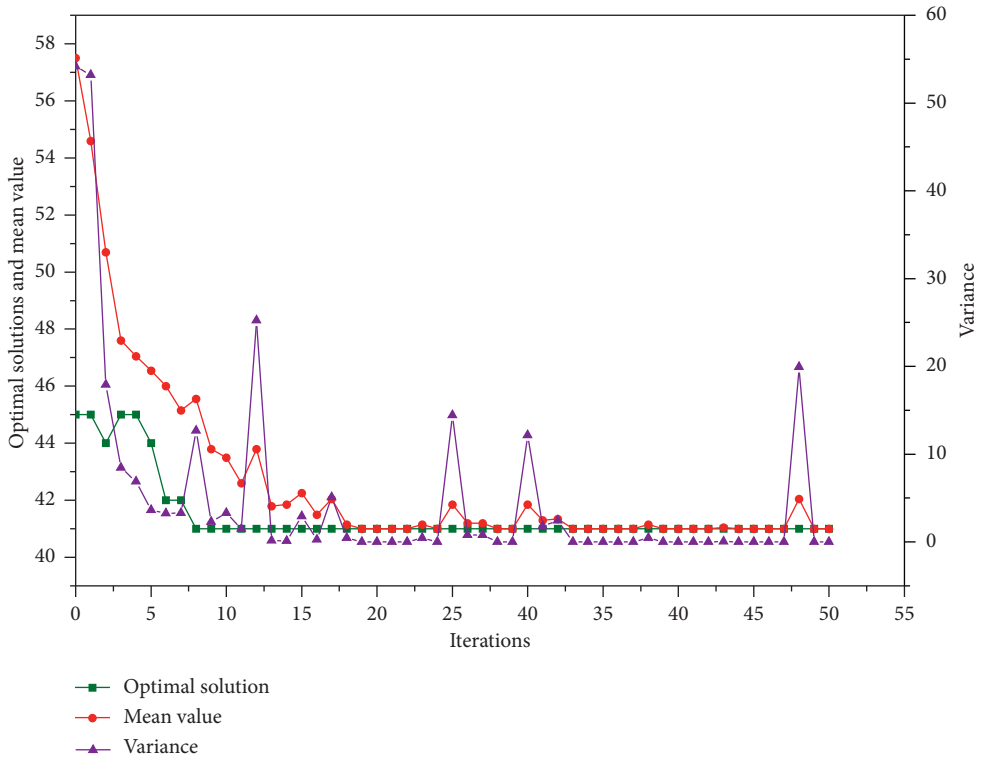


Figure 10: The population evolutionary process of Case 1.

average solution time is 9.15 s, and 2 different Gantt charts of the optimal scheduling are output. The reference of Xie [25] got the optimal solution 31. The reference of Xie [27] got the optimal solution 28. Figure 16 is the population evolutionary process of diffident algorithm for Case 2. Obviously, the results of the algorithm in this paper are better than those in the literature.

Figure 17 is the population distribution map of Case 2. It is obvious that the population approaches the optimal solution with the increase of iteration times. According to the population density, it shows that obvious population aggregation occurred in the 2nd, 4th, 6th, 7th, and 9th generations, and the aggregation solutions were 35, 34, 35, 31, and 29, respectively, which is the result of GA selection.

Figure 18 is the population evolutionary process of Case 2. The optimal solution was obtained in the 12th generation. With the change of population iteration times, the mean and variance of individual fitness also show good convergence. In the 12th generation, the variance suddenly increased,

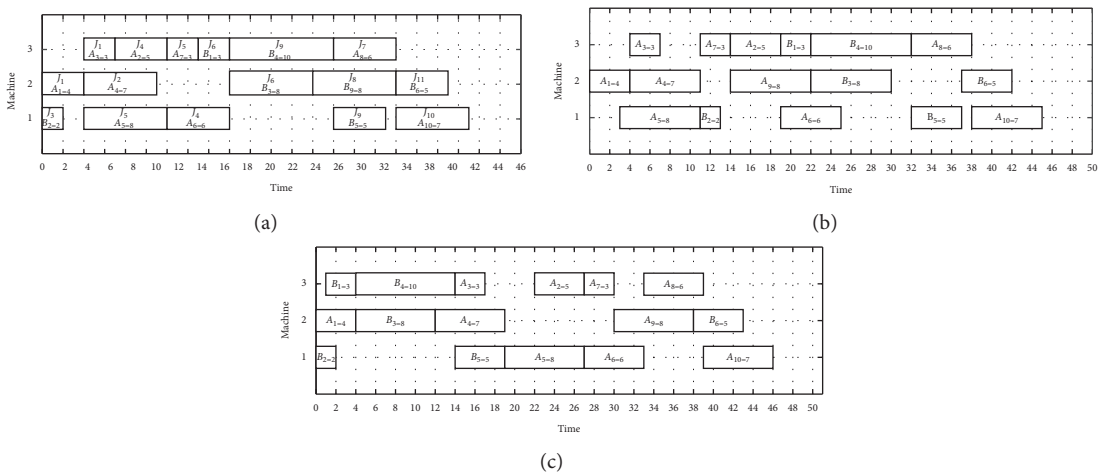


Figure 11: The Gantt chart of Case 1. (a) The minimum makespan is 41, (b) the minimum makespan is 45, and (c) the minimum makespan is 46.

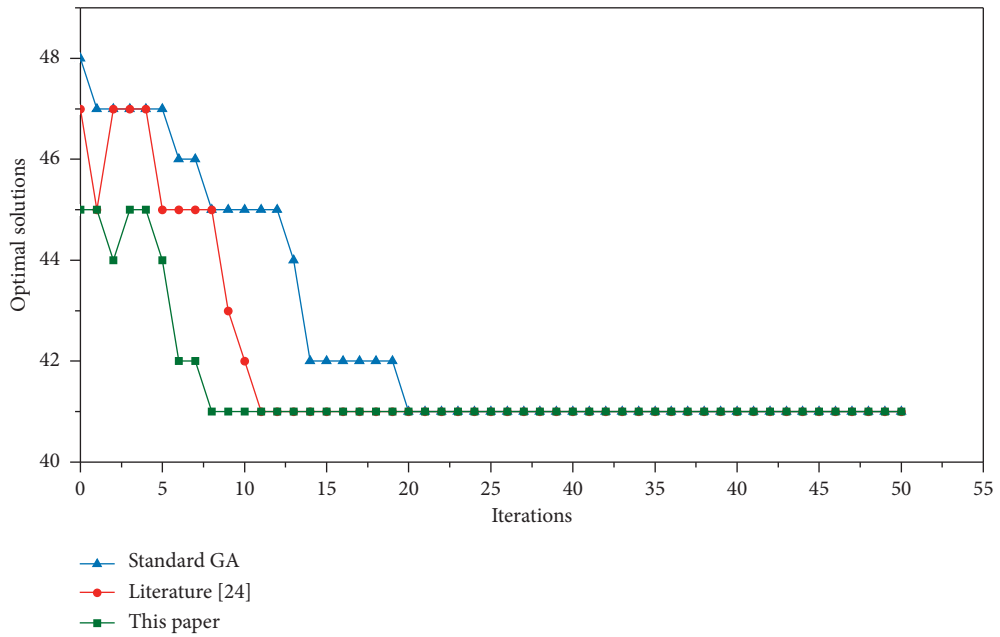


Figure 12: The population evolutionary process of diffident algorithm for Case 1.

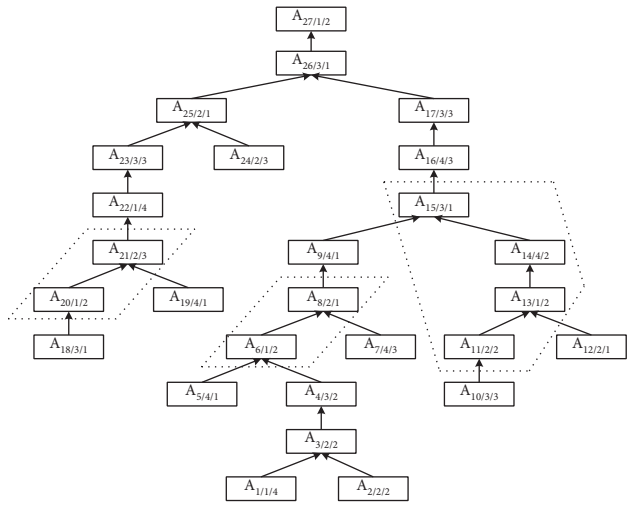


Figure 13: Operation tree of Case 2.

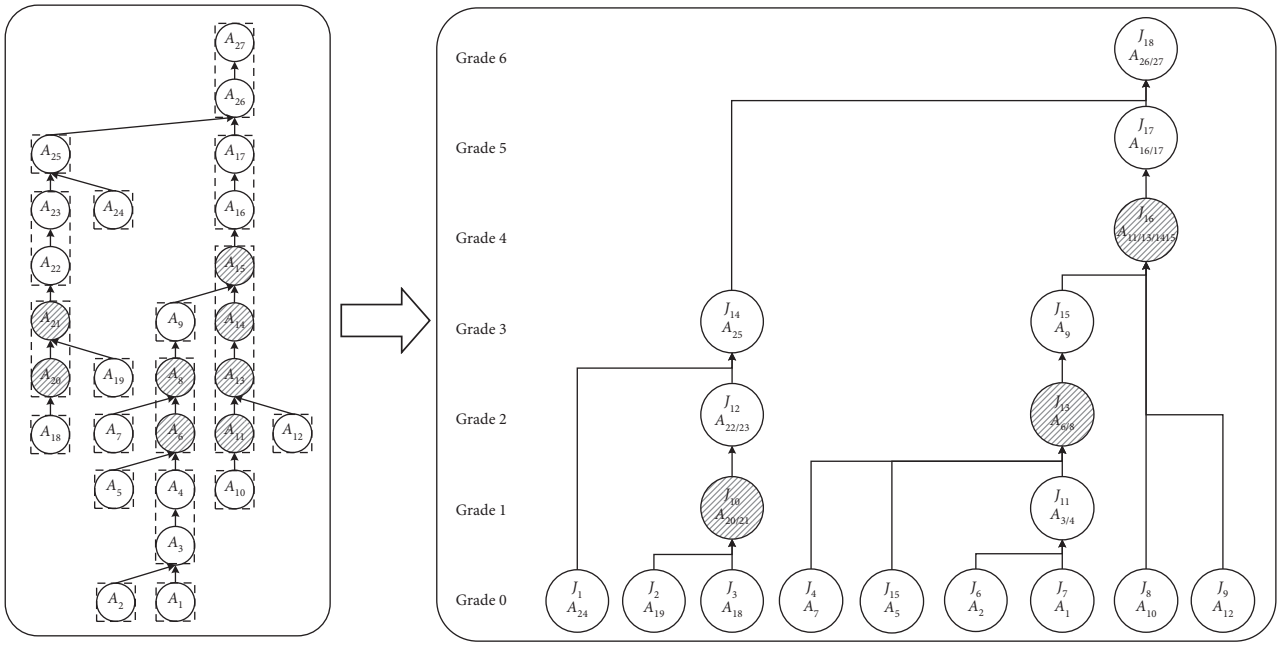
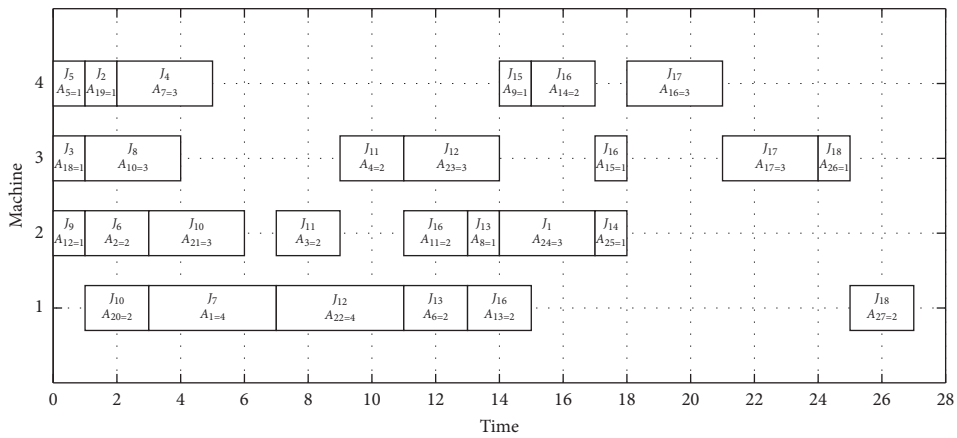
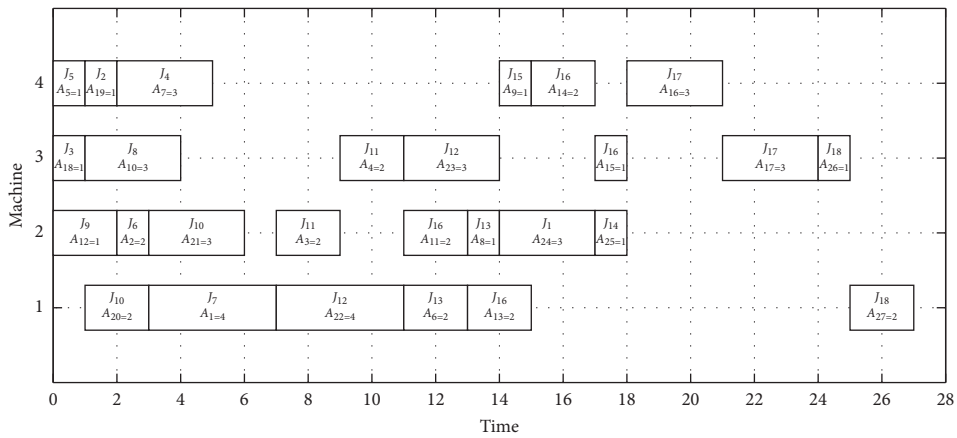


Figure 14: Pruning and grading result of Case 2.

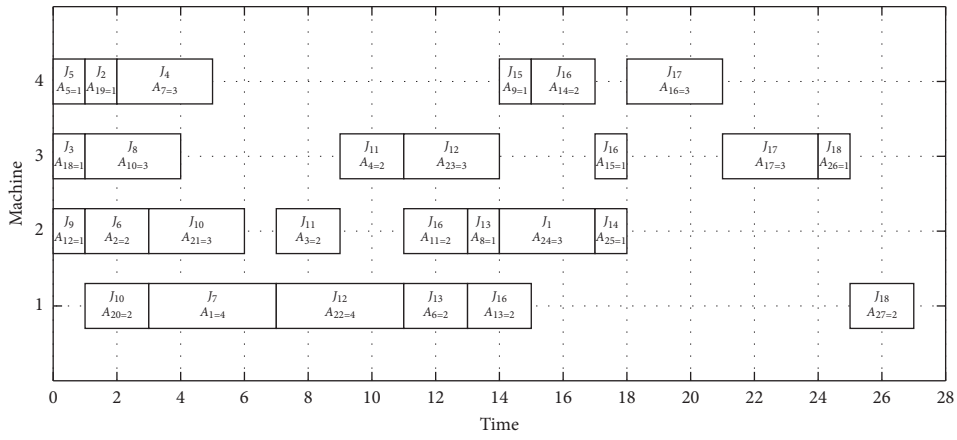


(a)

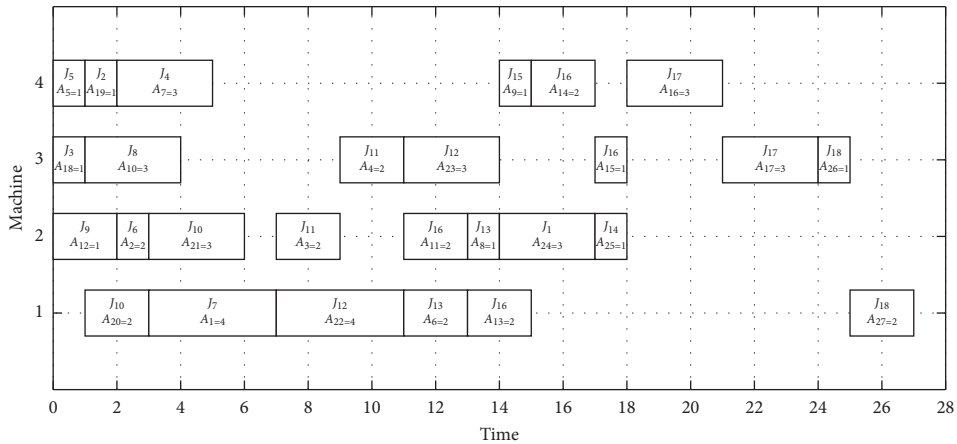


(b)

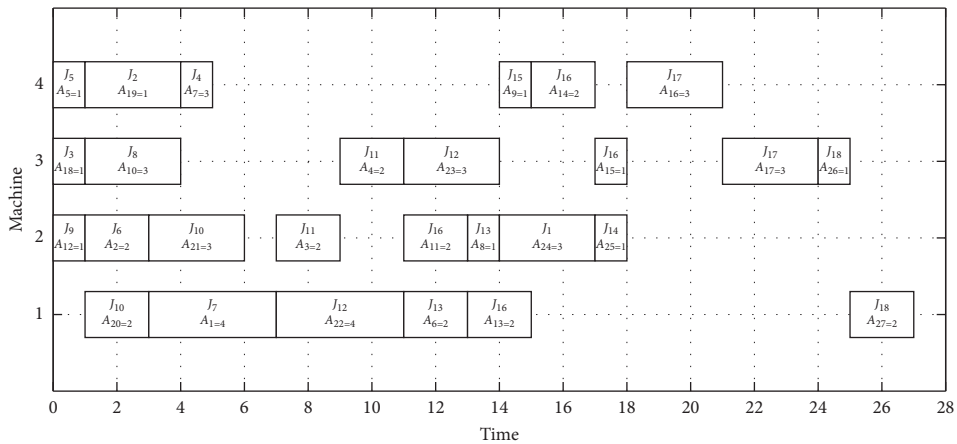
Figure 15: Continued.



(c)

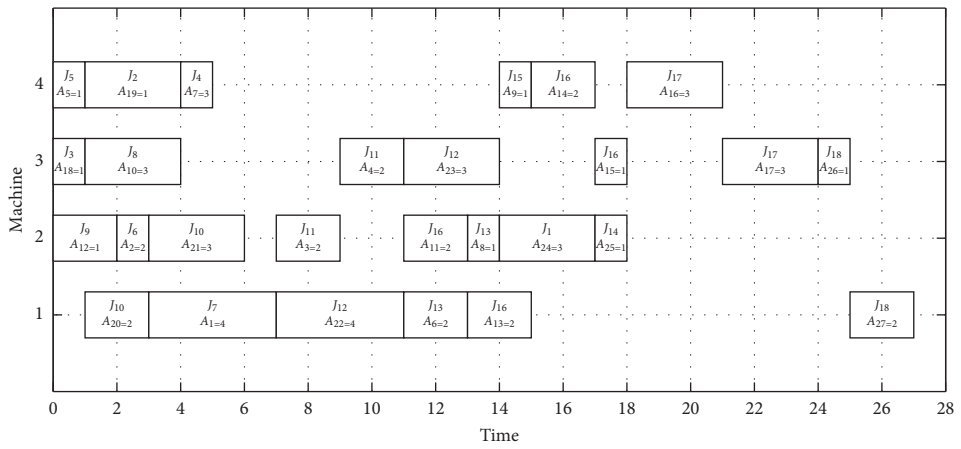


(d)



(e)

Figure 15: Continued.



(f)

Figure 15: The Gantt chart of Case 2. (a–f) The minimum makespan is 27.

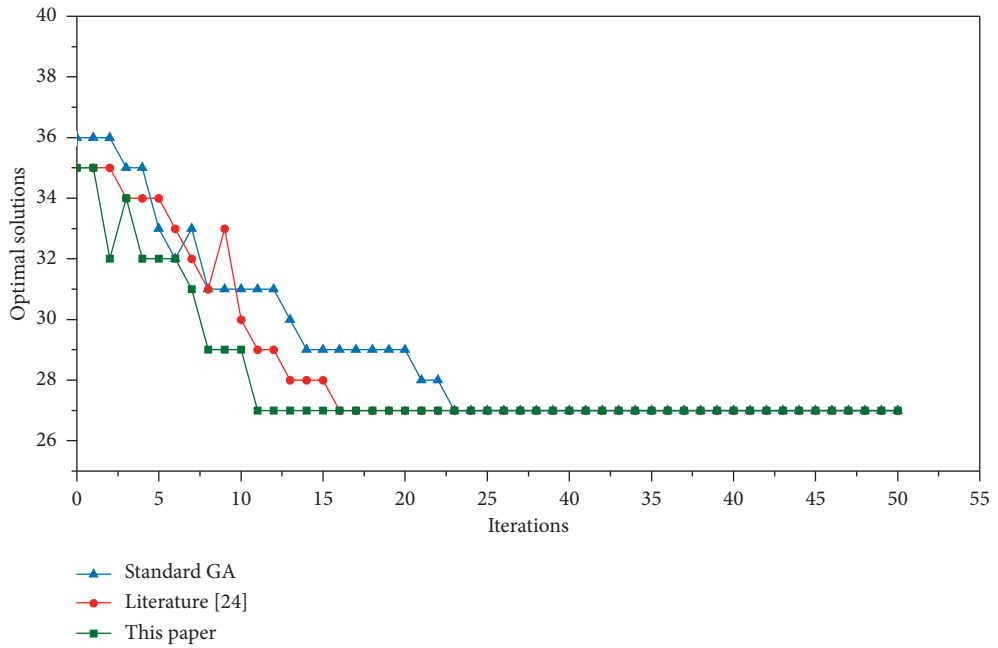


Figure 16: The population evolutionary process of diffident algorithm for Case 2.

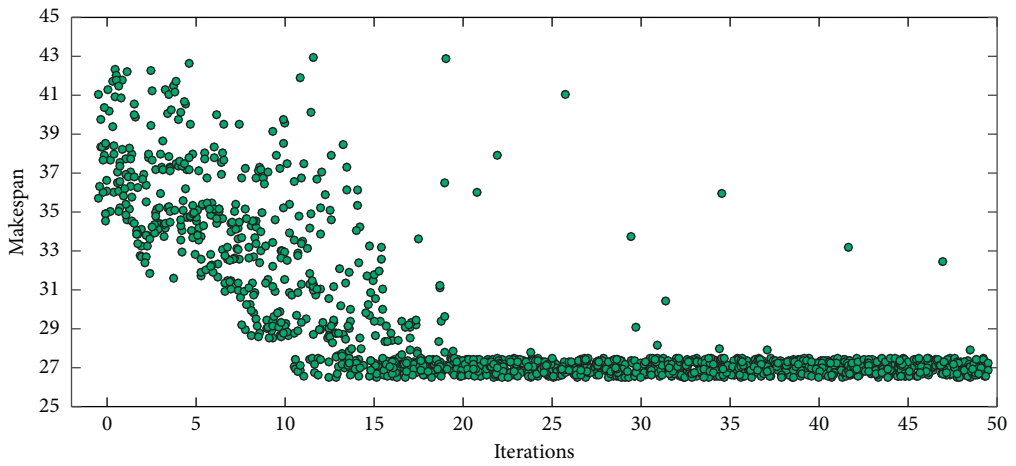


Figure 17: The population distribution map of Case 2.

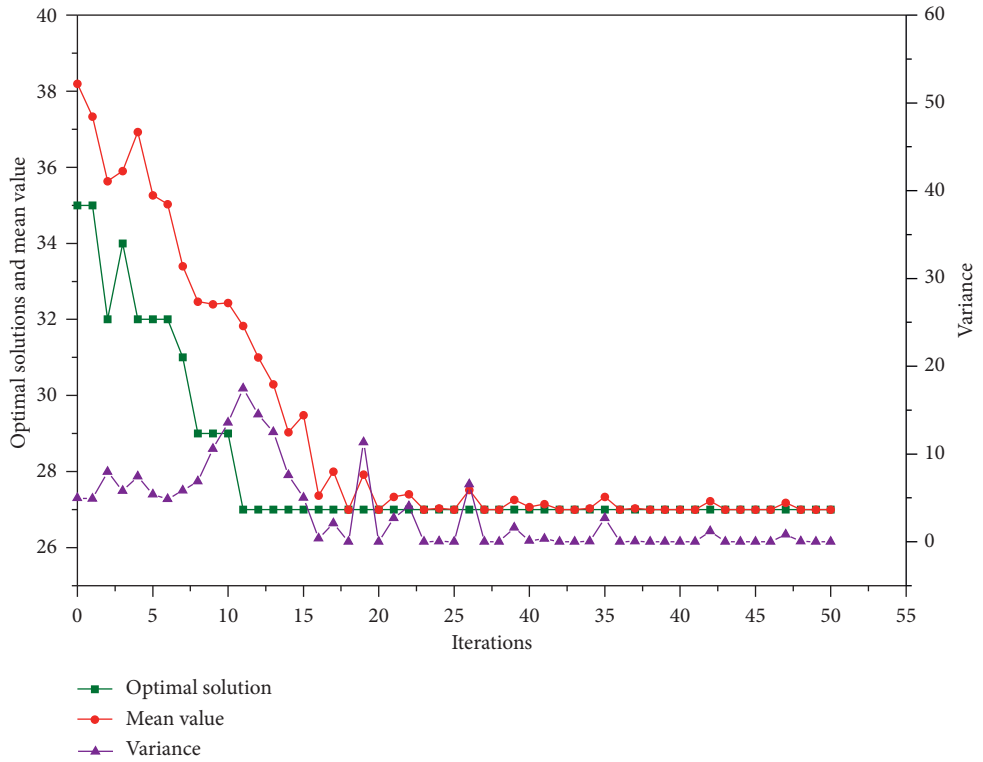


Figure 18: The population evolutionary process of Case 2.

which also represents that the individual diversity of the population suddenly became rich, which is beneficial to the global optimization.

5. Conclusion

Aiming at the complex multiproduct scheduling problem with 0-wait constraint, a hybrid algorithm based on GA and SA algorithm was studied. Based on the results of pruning and grading to the operation tree of complex multiproduct, the DSM with precedence constraints was established. Then, an initial population coding method based on DSM was proposed, which ensures the feasibility of the initial solution. At the same time, three strategies to optimize the initial population were proposed to improve the quality of the initial population for the situation of multiple operations in the same grade which need to be processed on the same machine, and they were more-task operation first, long-path operation first, and short-time operation first. For the hybrid algorithm, the specific process flow and the setting method of related parameters were carried out. For the infeasible solution produced in the crossover operation, the repair method was proposed. In the process of decoding, the chromosome genes were classified and the decoding for complex multiproduct scheduling problem with 0-wait constraint was realized through the analysis of its characteristics. The effectiveness of the proposed algorithm for complex multiproduct scheduling problem with 0-wait constraint is verified by the test of related examples in the existing literature.

For the general JSP, there are process constraints in the internal production process of workpiece. Different from the general JSP, the complex multiproduct scheduling problem has sequence constraints between workpieces, which is equivalent to a multilayer JSP. With the 0-wait constraint, the model of this problem can provide guidance for solving the scheduling requirements of small-batch special production. The DSM-based coding method proposed in this paper can also introduce other intelligent algorithms to solve the integrated product scheduling problem.

The neighborhood operators designed in this paper focus more on ensuring the feasibility of generating offspring individuals, which makes it unable to compete with the excellent neighborhood operators in traditional job-shop scheduling and reduces the search ability of the algorithm. Seeking or researching better neighborhood operators to improve the speed and accuracy of the algorithm is the direction worthy of further study in the future.

Data Availability

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

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

Acknowledgments

This work was supported by the Key Research and Development Plan of Shandong Province (2019GGX104102) and the Natural Science Foundation of Shandong Province (ZR2017MEE066).

References

[1] S. M. Johnson, "Optimal two- and three-stage production schedules with setup times included," *Naval Research Logistics Quarterly*, vol. 1, pp. 61–68, 1954.

[2] S. P. Bansal, "Minimizing the sum of completion times of n jobs over m machines in a flowshop—a branch and bound approach," *AIIE Transactions*, vol. 9, pp. 306–311, 1977.

[3] K. C. Ying, C. C. Lu, and S. W. Lin, "Improved exact methods for solving no-wait flowshop scheduling problems with due date constraints," *IEEE Access*, vol. 6, pp. 30702–30713, 2018.

[4] A. A. Ozolins, *New Exact Algorithm for No-Wait Job Shop Problem to Minimize Makespan*, Springer, Berlin, Heidelberg, Germany, 2020.

[5] J. Wu, G. D. Wu, and J. J. Wang, "Flexible job-shop scheduling problem based on hybrid ACO algorithm," *International Journal of Simulation Modelling*, vol. 16, pp. 497–505, 2017.

[6] H. Wei, S. Li, H. Jiang, J. Hu, and J. Hu, "Hybrid genetic simulated annealing algorithm for improved flow shop scheduling with makespan criterion," *Applied Sciences*, vol. 8, no. 12, p. 2621, 2018.

[7] F. Shi, S. Zhao, and Y. Meng, "Hybrid algorithm based on improved extended shifting Bottleneck procedure and GA for assembly job shop scheduling problem," *International Journal of Production Research*, vol. 58, pp. 2604–2625, 2020.

[8] M. S. Nagano, F. S. de Almeida, and H. H. Miyata, "An iterated Greedy algorithm for the no-wait flowshop scheduling problem to minimize makespan subject to total completion time," *Engineering Optimization*, vol. 4, pp. 1–19, 2020.

[9] P. Mudjihartono, R. Jiamthapthaksin, and T. Tanprasert, "Parallelized GA-PSO algorithm for solving job shop scheduling problem," in *Proceedings of the 2016 2nd International Conference on Science in Information Technology (ICSITech 2016)*, Balikpapan, Indonesia, 2017.

[10] Q. Liu, M. Zhan, F. O. Chekem et al., "A hybrid fruit fly algorithm for solving flexible job-shop scheduling to reduce manufacturing carbon footprint," *Journal of Cleaner Production*, vol. 168, pp. 668–678, 2017.

[11] A. Allahverdi, H. Aydilek, and A. Aydilek, "No-wait flowshop scheduling problem with separate setup times to minimize total tardiness subject to makespan," *Applied Mathematics and Computation*, vol. 365, Article ID 124688, 2020.

[12] H. Zhang, S. Liu, S. Moraca, and R. Ojstersek, "An effective use of hybrid metaheuristics algorithm for job shop scheduling problem," *International Journal of Simulation Modelling*, vol. 16, pp. 644–657, 2017.

[13] M. Nouiri, A. Bekrar, A. Jemai et al., "Two stage particle swarm optimization to solve the flexible job shop predictive scheduling problem considering possible machine breakdowns," *Computers and Industrial Engineering*, vol. 112, pp. 595–606, 2017.

[14] X. Wu, J. Peng, X. Xiao, and S. Wu, "An effective approach for the dual-resource flexible job shop scheduling problem considering loading and unloading," *Journal of Intelligent Manufacturing*, vol. 32, pp. 707–728, 2021.

[15] X. Huang, X. Zhang, S. M. Sardar, and C. A. Vega-Mejía, "An enhanced genetic algorithm with an innovative encoding strategy for flexible job-shop scheduling with operation and processing flexibility," *Journal of Industrial and Management Optimization*, vol. 16, pp. 2943–2969, 2020.

[16] D. Yüksel, M. F. Taşgetiren, L. Kandiller, and L. Gao, "An energy-efficient bi-objective no-wait permutation flowshop scheduling problem to minimize total tardiness and total energy consumption," *Computers and Industrial Engineering*, vol. 145, Article ID 106431, 2020.

[17] X. Wu and A. Che, "Energy-efficient no-wait permutation flow shop scheduling by adaptive multi-objective variable neighborhood search," *Omega*, vol. 94, Article ID 102117, 2020.

[18] Z. Wang, J. Zhang, and S. Yang, "An improved particle swarm optimization algorithm for dynamic job shop scheduling problems with random job arrivals," *Swarm and Evolutionary Computation*, vol. 51, Article ID 100594, 2019.

[19] J. Dong, H. Pan, C. Ye, and W. Tong, "No-wait two-stage flowshop problem with multi-task flexibility of the first machine," *Information Sciences*, vol. 544, pp. 25–38, 2021.

[20] Y. Xu, X. Li, C. Shi et al., "A metaheuristic for no-wait flowshops with variable processing times," in *Proceedings of the 2018 IEEE 22nd International Conference on Computer Supported Cooperative Work in Design*, pp. 479–484, Nanjing, China, May 2018.

[21] K. C. Ying and S. W. Lin, "Solving no-wait job-shop scheduling problems using a multi-start simulated annealing with bi-directional shift timetabling algorithm," *Computers and Industrial Engineering*, vol. 146, Article ID 106615, 2020.

[22] G. Deng, Q. Su, Z. Zhang et al., "A population-based iterated greedy algorithm for no-wait job shop scheduling with total flow time criterion," *Engineering Applications of Artificial Intelligence*, vol. 88, Article ID 103369, 2020.

[23] Q. Lei, W. Guo, and Y. Song, "Integrated scheduling algorithm based on an operation relationship matrix table for tree-structured products," *International Journal of Production Research*, vol. 56, pp. 5437–5456, 2018.

[24] W. Guo, Q. Lei, Y. Song, X. Lv, and L. Li, "Integrated scheduling algorithm of complex product with no-wait constraint based on virtual component," *Journal of Mechanical Engineering*, vol. 56, no. 4, pp. 262–273, 2020.

[25] Z. Xie, Z. Li, S. Hao, and G. Tan, "Study on complex product scheduling problem with no-wait constraint between operations," *Acta Automatica Sinica*, vol. 35, no. 7, pp. 983–989, 2009.

[26] Z. Xie, S. Liu, and P. Qiao, "Dynamic job-shop scheduling algorithm based on ACPM and BFSM," *Journal of Computer Research and Development*, vol. 40, no. 7, 2003.

[27] Z. Q. Xie, Y. Z. Teng, and J. Yang, "Integrated scheduling algorithm with no-wait constraint operation group," *Acta Automatica Sinica*, vol. 37, no. 3, pp. 371–379, 2011.

[28] Z. Xie, Y. Xin, and J. Yang, "No-wait integrated scheduling algorithm based on reversed order signal-driven," *Journal of Computer Research and Development*, vol. 50, no. 8, pp. 1710–1721, 2013.

[29] S. Zhao, Q. Han, and G. Wang, "Product comprehensive scheduling algorithm based on virtual component level division coding," *Computer Integrated Manufacturing Systems*, vol. 21, no. 9, pp. 2435–2445, 2015.

[30] K. C. Ying and S. W. Lin, "Minimizing makespan for no-wait flowshop scheduling problems with setup times," *Computers and Industrial Engineering*, vol. 121, pp. 73–81, 2018.

[31] Y. Wu, "A survey on population-based meta-heuristic algorithms for motion planning of aircraft," *Swarm and Evolutionary Computation*, vol. 62, Article ID 100844, 2021.

[32] Y. Wu, S. Wu, and X. Hu, "Cooperative path planning of UAVs UGVs for a persistent surveillance task in urban environments," *IEEE Internet of Things Journal*, vol. 8, no. 6, pp. 4906–4919, 2021.
[33] E. S. H. Hou, N. Ansari, and H. Ren, "A genetic algorithm for multiprocessor scheduling," *IEEE Transactions on Parallel and Distributed Systems*, vol. 5, no. 2, pp. 113–120, 1994.
[34] D. V. Steward, "Design structure system: a method for managing the design of complex systems," *IEEE Transactions on Engineering Management*, vol. 28, no. 3, pp. 71–74, 1981.
[35] C. Zhang, Y. Rao, X. Liu, and P. Li, "An improved genetic algorithm for the job shop scheduling problem," *China Mechanical Engineering*, vol. 15, no. 23, pp. 2149–2153, 2004.