Integrated scheduling algorithm based on the priority constraint table for complex products with tree structure

Yilong Gao1, Zhiqiang Xie1*, Xinyang Liu1, Wei Zhou1 and Xu Yu2

Abstract
Aiming at the existing intelligent optimization algorithms for solving the integrated scheduling problem of complex products with tree structure, there are problems of missing optimal solutions when designing encoding methods or generating infeasible offspring while designing evolutionary operators, an integrated scheduling algorithm based on the priority constraint table is proposed in this paper. A novel encoding method based on the dynamic priority constraint table is developed, which can guarantee the feasibility and completeness of the initial population individuals. For the legitimacy of the generated offspring individuals, two new different crossover and mutation methods are designed separately. The introduced evolutionary operators can avoid the detection and repairment of the infeasible individuals. An insertion-based greedy decoding method is also developed. In addition, based on the critical operations, a local search strategy is presented to enhance the search ability for the superior solutions. The feasibility and superiority of the proposed algorithm is verified by comparative experiments.

Keywords
Priority constraint table, tree-structured products, integrated scheduling algorithm, genetic algorithm, processing and assembly

Introduction
The production scheduling problem is a typical NP-hard problem, which usually treats processing1,2 and assembly3,4 as two independent stages. The final product is assembled by a series of components, and these components may be parts or sub-assemblies (SUAs) which have been previously assembled. At present, research on production scheduling problems mainly focuses on the manufacturing stage, such as the flow shop scheduling problem5,6 for mass production environments, and the job shop scheduling problem7,8 for multi-variety and small-batch production environment. Figure 1 gives three typical product structures, which show the relationship between parts or assembled components and products in a hierarchical way. In flat structure products, the final product is assembled from a series of parts. In tall structured products, there can be at most one assembly operation in each layer. In complex structure products, each layer can have multiple assembly operations. In this paper, we refer to the product with the first structure as a simple product and the product with the latter two structures as a complex product. With the increase of people’s demand for

---

Creative Commons CC BY: This article is distributed under the terms of the Creative Commons Attribution 4.0 License (https://creativecommons.org/licenses/by/4.0/) which permits any use, reproduction and distribution of the work without further permission provided the original work is attributed as specified on the SAGE and Open Access pages (https://us.sagepub.com/en-us/nam/open-access-at-sage).
mainly focuses on simple products. The literatures on the integrated scheduling of tree-structured products integrated scheduling problem. At present, the research of intelligent optimization algorithms for solving the constraints among operations are tree structure. There are machines for processing as well as machines for assembly. The research on the integrated scheduling of tree-structured products can shorten the manufacturing cycle of products and improve the competitiveness of enterprises.

Due to the characteristics of the NP-hard problem, there is no effective algorithms to solve the production scheduling problems in polynomial time. In literature, the authors noticed that the metaheuristic algorithms can effectively solve production scheduling problems and other combinatorial optimization problems. In the past few decades, for improving the manufacturing efficiency and competitiveness of enterprises, various kind of intelligent optimization algorithms are applied to deal with production scheduling problems, such as genetic algorithm, artificial bee colony algorithm, particle swarm optimization algorithm, teaching-learning-based optimization algorithm, water wave optimization algorithm, biogeography-based optimization algorithm, hybrid algorithm and so on. However, due to the existence of the inherent tree-structured constraint relationship of complex products, all existing encoding methods and evolutionary operators have been invalidated, which limits the application of intelligent optimization algorithms for solving the integrated scheduling problem. At present, the research on the integrated scheduling of tree-structured products mainly focuses on simple products. The literatures on the integrated scheduling of complex products are relatively few, and most of them are rule-based heuristic algorithms. For example, in literatures, the authors have systematically and comprehensively studied the integrated scheduling of complex tree-structured products, and they have developed lots of integrated scheduling algorithms, which are based on the processing operation tree model. But as the scale of the problem increases, the computation time of these algorithms cannot be guaranteed. For the intelligent optimization algorithm-based solutions, the authors have shown approaches based on genetic algorithm separately, but they all need to consider additional detecting and repairing work to avoid generating infeasible offspring individuals, which lead to the increase of computation time and the decrease of efficiency of the algorithm. In literature, the authors put forward a virtual component level division encoding method, and then solve the problem based on GA. This algorithm can solve the above problems, but its encoding method is flawed, that is, the optimal solution may be missed. In literature, the authors propose an integrated scheduling algorithm based on an operation relationship matrix table. The designed evolutionary operators focus more on guaranteeing the viability of the offspring individuals, thus, they are yet excellent evolutionary operators. In literature, the authors present a hybrid algorithm based on improved extended shifting bottleneck procedure and GA. Although the problems of imposition constraints and omission of solution spaces are avoided, the repair work for the infeasible solutions generated by genetic operators is still needed.

To solve the problem of previous metaheuristic-based integrated scheduling algorithms, in this paper, a novel integrated scheduling algorithm of complex products based on the priority constraint table is proposed. The designed encoding method can defeat the shortcomings of the encoding method in the previous literature that may omit the optimal solutions, and in the meantime, it can make the design of the subsequent evolutionary operator easier. For the crossover and mutation operators, two different new methods are presented to ensure the legitimacy of the generated offspring chromosomes. A simple but effective decoding method is given. In addition, based on the critical operation set of the optimal scheduling scheme, a local search strategy is presented. Finally, the performances of the introduced algorithm are verified by experiments.

**Problem description**

The differences between the integrated scheduling problem of complex products and traditional production scheduling problem are that the problem studied in this paper needs to consider the assembly order constraints among jobs, which makes the processing structure of products tree-like. Figure 2 is a processing operation tree of a specific product. In Figure 2, the operation is represented by a node, and each node is
composed of three parts, respectively representing the operation’s name, the processing machine and the corresponding time. The direction of the arrow shows the priority constraint relationship between operations. The problem studied in this paper can be described as follows: a single complex product consisting of \( n \) operations \( \{ O_i \}_{1 \leq i \leq n} \) needs to be processed on \( m \) machines. The available machine of operation \( O_i \) is \( \{ M_j \}_{1 \leq j \leq m} \subseteq \{ 1, \ldots, m \} \) and \( M_i \neq \emptyset \). \( p_{ik} \) is the required processing time of operation \( O_i \) on machine \( k \in M_i \), \( s_i \) is the start time of operation \( O_i \) while \( c_i \) is the completion time of operation \( O_i \). \( x_{ij} = 1 \) indicates that operation \( O_j \) is a technology successor of operation \( O_i \); \( y_{ik} = 1 \) indicates that operation \( O_i \) is processed on machine \( k \in M_i \); \( z_{ij} = 1 \) denotes that operation \( O_i \) and \( O_j \) are assigned to the same machine for processing, and the operation \( O_i \) is the machine predecessor of \( O_j \), that is, the operation \( O_j \) is started after \( O_i \) is completed.

The objective function

\[
\min \{ \max( c_i ) \}_{i = 1, \ldots, n},
\]

is subject to

\[
\sum_{k=1}^{m} y_{ik} = 1, \quad i = 1, \ldots, n \quad (2)
\]

\[
p_i = \sum_{k=1}^{m} y_{ik} p_{ik}, \quad i = 1, \ldots, n \quad (3)
\]

\[
s_i \geq \max \{ c_l \}_{l = 1, \ldots, n \text{ and } x_{ik} = 1}, \quad i = 1, \ldots, n \quad (4)
\]

\[
c_i = s_i + p_i, \quad i = 1, \ldots, n \quad (5)
\]

\[
z_{ij} + z_{jk} \geq y_{ik} + y_{jk} - 1, \quad k \in M_i \cap M_j \quad (6)
\]

\[
s_j \geq s_i + (1 - z_{ij} L) c_l, \quad M_i \cap M_j \neq \emptyset \quad (7)
\]

\[
s_i \geq 0, \quad i = 1, \ldots, n \quad (8)
\]

Among them, equation (1) is the objective function of the problem, which is to minimize the completion time of the product. Equation (2) means that each operation should be assigned to only one machine for processing. Equation (3) denotes the processing time of operation \( O_i \). Constraint (4) indicates that \( O_i \) can only be processed after all its technology predecessors having finished. Equation (5) denotes the completion time of operation \( O_i \). Constraints (6) and (7) indicate that for two consecutive operations assigned to the same machine, the start time of the latter operation cannot be earlier than the completion time of the former operation. \( L \) represents a sufficiently large positive constant. Constraint (8) indicates that the start time must have general significance, that is, the start time of operations must be greater than or equal to 0.

**Algorithm design**

As an evolutionary search algorithm, the genetic algorithm has shown good effectiveness in solving traditional flow shop scheduling problem and job shop scheduling problem. Therefore, for the integrated scheduling problem of complex products with tree structure, a GA-based integrated scheduling algorithm is presented in this section. In order to facilitate the design of the subsequent strategies, we simplify the processing operation tree and only describe the priority constraint relationships among operations. Then its corresponding priority constraint graph (PCG) can be obtained. Figure 3 is a priority constraint graph of the product shown in Figure 2. The priority constraint graph is essentially a directed acyclic graph, and the edge indicates the priority relationship between two operations. For example, \( A17 \rightarrow A12 \) in Figure 3, that is, \( A17 \preceq A12 \) indicates operation \( A12 \) cannot start...
processing until A17 is finished. For the convenience of
derivation, we call A17 a technology predecessor of
A12, and A12 as a technology successor of A17. For
any two operations processed on the same machine, if
operation $O_j$ is processed before operation $O_i$, then we
call $O_j$ a machine predecessor of $O_i$.

**Establishment of the priority constraint table**

Due to the existence of priority sequence constraints,
traditional genetic algorithms cannot be directly used to
solve the studied problem. In order to avoid generating
infeasible individuals during population evolution, in
this section, the process of establishing the priority con-
straint table will be given first. The following symbols
and variables are introduced and explained as follows:

- **PCT**: The priority constraint table of a single com-
  plex product.
- $m_j$: The element values in the $j$th column of the
  PCT.
- $m_i$: The element values in the $i$th row of the PCT.
- $m_{ij}$: The element value in the $i$th row and $j$th column
  of the PCT.

Thereinto, the element value of $m_{ij}$ in PCT is defined
as follows:

$$m_{ij} = \begin{cases} 
-1, & \text{If operation } O_j \text{ is the starting point of} \\
& \text{the arrow } e_i \text{ in } PCG. \\
1, & \text{If operation } O_j \text{ is the end point of}\\n& \text{the arrow } e_i \text{ in } PCG. \\
0, & \text{Otherwise} 
\end{cases}$$

According to the above definition and the priority con-
straint graph, the priority constraint table of a complex
product can be established. Table 1 is a priority con-
straint table of the product shown in Figure 3. The
priority constraint table indirectly expresses the
dependence relationship among operations. It is essen-
tially a dynamic correlation matrix. In a PCT, each
column records the operation information and each row
shows the priority constraint relationship between two
operations. The PCT not only facilitates the under-
standing of sequence constraints among operations,
but also facilitates the entry of product information.
Therefore, based on the priority constraint table, this
paper will present new encoding method and evolution-
ary operators subsequently.

**The encoding method based on the dynamic PCT**

In this paper, we adopt the operation-based encoding
strategy, that is, the integer on each gene position rep-
resents the operation’s number of the complex product.
In order to ensure the operation genes of the chromo-
somes satisfy the priority constraint defined by PCG, a
novel encoding scheme based on the dynamic priority
constraint table is proposed. The main idea is: without
considering the use of machines, identify the priority
constraint table and obtain the current schedulable
operation set. Then a schedulable operation is ran-
domly determined for processing. The specific steps of
the designed encoding method are as follows:

1. **Step 1**: Identify the priority constraint table $PCT$, and gain the current schedulable operation set $S$.

   The specific steps are: According to the rules for establishing the priority constraint table, if the non-
   zero element of $m_j$ in PCT is only $-1$, then there is
   no edge pointing to the operation node $O_j$. Hence,
no operations needs to be processed before $O_j$ is finished. So operation $O_j$ is a schedulable operation at the current time. Therefore, the current schedulable operation set $S$ can be gotten, by traversing each column in $PCT$ and finding out the corresponding operation of the $j$th column when the non-zero element in $m_j$ is only $-1$.

**Step 2:** Randomly select an operation $O_i$ from the schedulable operation set $S$, arrange the operation $O_i$ and update current $PCT$. The specific steps are: If the currently selected operation is $O_i$, then $O_i$ is processed, and it will be moved from the schedulable operation set to the scheduled operation set. At the same time, if the value of $m_{ji}$ is $-1$, then set $m_{ji} = 0$, that is, the sequence constraint relationship between operation $O_i$ and $O_j$ needs to be released.

**Step 3:** Repeat **Step 1**, and if the current set $S$ is empty, then the encoding process is complete; otherwise, jump to **Step 2**.

Through the above-mentioned encoding method, a chromosome which fully meets the priority constraint relationships among operations can be obtained. Figure 4 is the chromosome schematic diagram obtained by applying the above encoding method to the product shown in Figure 2.

**The decoding method based on greedy insertion**

Based on the characteristics of the integrated scheduling problem, in order to make all operations start processing as early as possible, an insertion-based greedy decoding method is presented. The specific decoding steps are as follows:

**Step 1:** Initialize the pre-start time $p_{s_i}$, the start time $s_i$, and the completion time $c_i$ of operation $O_i$ to 0, where $1 \leq i \leq n$.

**Step 2:** Read an operation from the chromosome in order from left to right. Assume that the current operation is $O_i$, the corresponding machine is $M_k$, and the required time is $p_{i_k}$.

**Step 3:** For operation $O_i$, calculate its actual start time and completion time. The idle period $[t_i, t_e]$ on machine $M_k$ is acquired from front to back, and determine whether the current idle period is sufficient to process operation $O_i$, that is, if $\max(p_{s_i}, t_i) + p_{i_k} \leq t_e$, then set $s_i$ to $\max(p_{s_i}, t_i)$. Otherwise, the next idle period is checked. If operation $O_i$ can’t be processed in all idle periods of the machine $M_k$, then put it at the end of machine $M_k$, that is, set $s_i$ to $\max(p_{s_i}, C_k)$, where $C_k$ represents the completion time of the last operation which is processed on machine $M_k$. Then the completion time of operation $O_i$ is $c_i = s_i + p_{i_k}$.

**Step 4:** Update the pre-start time of the affected operations. Read the priority constraint table, obtain the technology successor of operation $O_i$, assuming the current technology successor is $O_j$, then the pre-start time of operation $O_j$ is $p_{s_j} = \max\{p_{s_j}, c_i\}$.

**Step 5:** If all operations has been processed, then the decoding process is finished; otherwise, jump to **Step 2**.

Through the above decoding method, a specific individual can be decoded into a scheduling scheme. Figure 5 is a specific scheme corresponding to the individual shown in Figure 4.
Selection method

In order to make the traits of good individuals inherited to the offspring with greater probability, the algorithm adopts the roulette selection method, supplemented by elite retention strategy.

\[ f_i = 1 - \frac{MS_i}{\sum_{i=1}^{n} MS_i} \]

is the fitness value of chromosome \( i \), where \( MS_i \) represents the corresponding makespan of chromosome \( i \).

Crossover operators

In genetic algorithm, crossover is an important part to ensure that the population evolves toward promising regions, which determines the global search ability for the optimal solutions. Due to the more complex priority constraints of the integrated scheduling problem, the existing crossover methods are invalid. Therefore, in order to ensure that after crossover process, operations of the offspring individuals can still meet the priority sequence constraints, and in the meantime, the offspring individuals can inherit the superior traits of the parent individuals, the following two crossover methods are designed.

1. The designed two-point crossover method

   **Step 1:** In the paired parental chromosomes \( V_1 \) and \( V_2 \), the starting and ending position of the recombinant gene string (the positions selected are the same in the two parent chromosomes) are determined randomly, and ensure that the number of genes in the recombinant gene string is greater than 1 and not equal to the length of the entire chromosome.

   **Step 2:** Generate the offspring individual \( V_1' \). The recombination position of \( V_1' \) is determined according to the starting and ending position of the recombinant gene string determined in **Step 1**. The recombinant positions are filled by the genes of the recombination gene string determined by \( V_1 \), and the arrangement order is the order in which they appear in \( V_2 \). The remaining parts of \( V_1' \) are the same as that of \( V_1 \). The generation procedure of offspring individual \( V_1' \) is shown in Figure 6(a).

   **Step 3:** Generate the offspring individual \( V_2' \). The recombination position of \( V_2' \) is determined according to the starting and ending position of the recombinant gene string determined in **Step 1**. The recombinant positions are filled by the genes of the recombination gene string determined by \( V_2 \), and the arrangement order is the order in which they appear in \( V_1 \). The remaining parts of \( V_2' \) are the same as that of \( V_2 \). The generation procedure of offspring individual \( V_2' \) is shown in Figure 6(b).

2. The designed multi-point crossover method

   **Step 1:** In the paired parental chromosomes \( V_1 \) and \( V_2 \), several recombinant genes are determined.

![Figure 6. The designed two-point crossover method: (a) generation of offspring chromosome \( V_1' \) and (b) generation of offspring chromosome \( V_2' \).](image-url)
stochastically, and positions of these genes may be discontinuous, but the positions selected of the two parent chromosomes are the same.

**Step 2:** Generate the offspring individual $V_1'$. According to the position of the recombinant genes determined in **Step 1**, the first consecutive recombinant gene string of $V_1$ is obtained as the current recombination string. Obtain the starting and ending position of the current recombinant string in $V_1$, and the recombination position of $V_1'$ is determined according to the starting and ending position of the current recombinant string. The recombinant positions are filled by the genes of the current recombinant string, and the arrangement order is the order in which they appear in $V_2$. Update the current recombinant string by the next consecutive recombinant gene string in $V_1$, and perform the recombination process in the above way until all recombinant genes in $V_1$ are recombined. The remaining genes of $V_1'$ are identical to the remaining genes of $V_1$. The generation procedure of offspring individual $V_1'$ is shown in Figure 7(a).

**Step 3:** Generate the offspring individual $V_2'$. According to the position of the recombinant genes determined in **Step 1**, the first consecutive recombinant gene string of $V_2$ is obtained as the current recombination string. Obtain the starting and ending position of the current recombinant string in $V_2$, and the recombination position of $V_2'$ is determined according to the starting and ending position of the current recombinant string. The recombinant positions are filled by the genes of the current recombinant string, and the arrangement order is the order in which they appear in $V_1$. Update the current recombinant string by the next consecutive recombinant gene string in $V_2$, and perform the recombination process in the above way until all recombinant genes in $V_2$ are recombined. The remaining genes of $V_2'$ are identical to the remaining genes of $V_2$. The generation procedure of offspring individual $V_2'$ is shown in Figure 7(b).

(3) Analysis of the designed crossover methods

Through simple analysis, it can be found that the above two crossover methods generate the offspring individuals by filling the crossover positions with the selected genes in the order in which they appear in the paired parent chromosomes. The genes of the selected gene string satisfy the priority constraints in the paired chromosomes, so the generated offspring individuals must be legitimate. Assuming that $O_i$ and $O_j$ are any two adjacent operations on a legitimate paired chromosome, there may be two cases of constraints between $O_i$ and $O_j$. One is that there is no constraint relationship between $O_i$ and $O_j$, and the other is that there is a priority constraint between $O_i$ and $O_j$. In the first case,
regardless of the order in which operation \( O_i \) and \( O_j \) appear in the other legal paired chromosome, operation \( O_i \) and \( O_j \) satisfy the constraint in the generated offspring individuals. In the second case, since both paired parental chromosomes are legal chromosomes, the order of operation \( O_i \) and \( O_j \) in the two parental chromosomes must be the same. Therefore, operation \( O_i \) and \( O_j \) are necessarily satisfied in the generated offspring individuals. When the number of genes in the selected gene string is greater than 2, we can continuously accumulate two genes and another gene, then the legality can be proved. The designed multi-point crossover method can be regarded as the continuous accumulation of the designed two-point crossover method, and the legitimacy of the generated offspring individuals can also be easily proved. Therefore, the above two crossover methods can ensure that the generated descendants satisfy the constraints of the problem, that is, no infeasible individual will be generated after performing the above two crossover operators.

Mutation operators
Since the existing mutation methods cannot be directly applied to treat the integrated scheduling problems, in order to ensure the offspring individuals can also satisfy the priority constraints after performing mutation operator, based on the priority constraint table, two new mutation methods are designed.

(1) The designed insertion-based mutation method

The main idea of the designed mutation method is: Firstly, randomly determine a gene in the parent chromosome \( V \). Then, by identifying the priority constraint table, obtain the technology predecessors and the technology successors of the selected gene operation, and mark their position in the parent chromosome \( V \). Finally, insert the selected gene into any position before the technology successor and after the position of the technology predecessor which is closest to the selected gene. If there is no technology predecessor, i.e. the selected operation is a leaf-node operation, and then randomly insert the selected operation into any position before the technology successor. If the selected operation is a root-node operation or the positions of the technology predecessor operation, the selected operation and the technology successor are consecutive, then another mutation gene operation is randomly selected. The procedure for performing the above mutation operation on a chromosome is shown in Figure 8.

(2) The designed scramble mutation method

Step 1: In the parent chromosome \( V \), randomly determine the starting and ending position of the mutant gene string, and ensure that the number of genes in the mutant gene string is greater than 1 and not equal to the length of the entire chromosome. Step 2: Add the genes of the mutant gene string to the gene stack, so that the operation genes in the same gene stack meet the priority constraints, and there is no constraint relationship among operations of different gene stacks. The specific procedures are as follows: obtain a gene from the mutated gene string in order from back to front, assuming currently it’s \( O_i \). Identify the priority constraint table and determine whether there is a priority constraint between the operations after \( O_i \) in mutant gene string and operation \( O_j \). If there is an operation, assuming it’s \( O_j \), have a priority constraint with \( O_i \). Then the current stacking operation gene \( O_i \) is moved into the gene stack where \( O_j \) is located, that is, \( O_i \) is the new top element of that gene stack. Otherwise, a new gene stack is opened up and the current operation gene is put into the new stack. Repeat the above steps until all the mutant genes have been stacked. If there is only one gene stack after performing the above operations, jump to Step 1; otherwise, jump to Step 3. Step 3: Randomly pop genes out of the gene stacks to generate the mutated gene substrings. The specific procedures are as follows: randomly select a gene stack and determine whether the gene stack is
empty. If it is empty, then randomly select another gene stack; otherwise, pop the top element of the selected stack as the mutation gene of the current mutation position. Repeat the above processes and obtain the gene for the next mutation position until all gene stacks are empty.

**Step 4:** The generated offspring chromosome $V_1$ can be obtained by putting the mutated gene string into the mutation site. It can be seen that the offspring individuals completely satisfy the priority constraints. The procedure for performing the above mutation operator on a chromosome is shown in Figure 9.

(3) Analysis of the developed mutation methods

It is not difficult to find that the above two mutation methods are based on the idea of the designed encoding method and its encoding principle in this paper. Therefore, after the mutation process, the offspring population will not contain infeasible individuals. In addition, even if the selected mutation gene or mutation gene sequence is identical, the generated offspring individuals may not be unique.

**The designed local search strategy**

After decoding an individual into a specific scheduling scheme, we can find one or more paths from the last processing operation to the initial moment. The operations on the path are closely arranged, and its length value is equal to the completion time of the current scheme. We define such a path as a critical path, and call the operation on the critical path as a critical operation. For the scheduling scheme described in Figure 5, Figure 9. The designed scramble mutation method.
the critical path is shown in the dashed box in Figure 10, we can find that the operations \( A_1, A_4, A_{10}, A_7, A_9, A_{17}, \) and \( A_{18} \) constitute a critical path. Since the total processing time of the operations on the critical path is equal to the completion time of the current individual, it is possible to shorten the product completion time only by adjusting the operation on the critical path to convert some of the critical operation into non-critical operations. Therefore, in order to enhance the search ability of the algorithm for the optimal solution, we implement a local search strategy for the scheduling scheme by performing a series of perturbations on the critical operations. In order to reduce the execution time of the algorithm, in this paper, we only perform the local search strategy on the schemes corresponding to the best individuals of the current generation population. The specific steps of the local search strategy based on the critical operation moving forward described in this paper are as follows.

**Step 1**: Obtain the critical operation set of the current scheduling scheme.
**Step 2**: Determine whether the critical operation set is empty, if it is empty, the local search process is finished; otherwise, read an operation from the critical operation set in turn, assuming that the current operation is \( O_c \).
**Step 3**: Obtain the machine predecessor set of \( O_c \) from the start time of \( O_c \) to the initial processing moment. Read a machine predecessor from the machine predecessor set in turn, assuming the current operation is \( O_m \).
**Step 4**: Determine whether \( O_c \) can be processed at the start time of \( O_m \), that is, whether the maximum completion time of the technology predecessors of \( O_c \) is less than or equal to the start time of \( O_m \). If it is, jump to Step 3; otherwise jump to Step 2.
**Step 5**: Put \( O_c \) before \( O_m \) for processing, and update the processing information of the operations affected by \( O_c \) and \( O_m \). Determine whether the new makespan is better than the old makespan. If it is, replace the old scheme with the new scheme and jump to Step 1; otherwise, jump to Step 3.

Through the above steps, we can transform some critical operations of the original scheduling scheme into non-critical operations, thereby achieving the goal of shortening the product completion time. For example, in the critical path shown in Figure 10, when judging whether the critical operation \( A_{10} \) can be moved forward, the technology predecessor of \( A_{10} \) is \( A_{16} \), and the machine predecessor of \( A_{10} \) is \( A_7 \). The completion time of \( A_{16} \) is less than the start time of \( A_7 \), which meets the conditions for moving forward. Therefore, \( A_{10} \) can be processed before \( A_7 \). After updating the processing information of the affected operations, the start time of \( A_2 \) remains unchanged, the start time of \( A_4 \) will be advanced and converted into a non-critical operation, and the start time of \( A_1 \) also be advanced. Finally, the total processing time is reduced, and the new scheduling scheme at this moment is shown in Figure 11. The new makespan (170) is better than the old makespan (175), and the new critical path is shown in the dashed box in Figure 11. It can be seen that the above-mentioned steps can further enhance the algorithm’s optimal solution search capability.

**Procedure of the proposed algorithm**

The overall flowchart of the designed integrated scheduling algorithm for complex products is shown in Figure 12.

**Computational experiments**

At present, there are huge number of benchmarks for traditional job shop scheduling problems, but no benchmark has been found for the integrated scheduling problem of complex products. Most of the previous integrated scheduling algorithms are rule-based
algorithms, and the test instances are all single specific instances. In order to fully test the performance of the proposed algorithm compared with the previous algorithms, we randomly generate 100 instances. Each instance is randomly generated by the following parameters: the number of operations is \([50,100]\), the number of machines is \([4,8]\), the processing time is \([1,20]\), and the number of processing operation tree layers is \([2, n/2]\), where \(n\) is the number of operations for current instance. The comparison algorithms are the product comprehensive scheduling algorithm based on virtual component level division coding (CSA_VCLDC),\(^{26}\) the integrated scheduling algorithm based on an operation relationship matrix table for tree-structured products (ISA_ORMT),\(^{27}\) the hybrid algorithm based on improved extended shifting bottleneck procedure and GA (ISA_HGA),\(^{28}\) and the algorithm ISA_PCT proposed in this paper. All algorithms are programmed and implemented in MATLAB R2016a and run on a computer with Win10, 64-bit operating system, Intel (R) Core(TM) i7-4810MQ CPU @2.80GHz, and 16G memory. Each algorithm is run 30 times independently for each instance.

**Parameters setting**

In order to analyze the influence of the parameters on the performance of ISA_PCT, we adopted the Taguchi method of design-of-experiment method (DOE)\(^{29}\) to determine the parameters of the algorithm proposed in this paper. The specific complex product from literature\(^{25}\) is used for testing. The instance is composed of the classic job shop standard test problem FT10, and its process assembly diagram is shown in Figure 13. There are four key parameters of the algorithm namely, population size \((P_s)\), crossover probability \((P_c)\), mutation probability \((P_m)\), \(G\) (If the algorithm can not improve the current solution for \(G\) (a given number) iterations, then the algorithm terminates.). We use four factor levels and the combination of different parameters is shown in Table 2. A orthogonal array \(L_{16}(4^4)\) is chosen. For each combination of parameters, the algorithm
runs independently 30 times and takes the average makespan obtained as the experimental index. The orthogonal array and the analysis of experimental results are shown in Table 3. The influence trend of each parameter on the experimental index is shown in Figure 14.

It can be seen from Table 3 and Figure 14 that: first, the parameter \( P_s \) has the greatest influence on the performance of \( ISA_PCT \). If \( P_s \) is too small, the population cannot effectively represent the distribution space of the problem solution. If \( P_s \) is too large, the execution time of the algorithm will increase. Second, the parameter that greatly affects the performance of the algorithm is \( G \). If \( G \) is too small, it will cause the algorithm to search the problem solution insufficiently, but if \( G \) is too large, it will increase the algorithm's search time. Third, parameter that affects the performance of the algorithm is \( P_m \). If \( P_m \) is too small, the algorithm tends to converge prematurely. If \( P_m \) is too large, the algorithm becomes a random search, which affects the convergence of the algorithm. The parameter that has the least impact on the performance of the algorithm is \( P_c \). If \( P_c \) is too small, the algorithm's ability to search the optimal solution will be reduced. If \( P_c \) is too large, the individual's excellent characteristics will be frequently destroyed. Based on the above analysis, four key parameters of the \( ISA_PCT \) are set as follows:

\[
P_s = \frac{n^4}{C_3}, \quad G = 100, \quad P_c = 0.7, \quad P_m = 0.3.
\]

In addition, the parameters of \( CSA_VCLDC, ISA_ORMT \) and \( ISA_HGA \) are set to the same as those of \( ISA_PCT \).

### Table 2. The parameter values of different factor levels.

| Parameters | Factor level |
|------------|--------------|
| \( P_s \)  | \( n^1 \)    | \( n^2 \)    | \( n^3 \)    | \( n^4 \)    |
| \( G \)     | 50           | 100          | 150          | 200          |
| \( P_c \)   | 0.6          | 0.7          | 0.8          | 0.9          |
| \( P_m \)   | 0.1          | 0.2          | 0.3          | 0.4          |

### Table 3. The orthogonal results of different factor levels.

| Experiment number | Factor level | Index |
|------------------|--------------|-------|
| \( P_s \)        | \( G \)      | \( P_c \) | \( P_m \) | Makespan average |
| 1                | 1            | 1      | 1          | 1            | 2023.2 |
| 2                | 1            | 2      | 2          | 2            | 2000.4 |
| 3                | 1            | 3      | 3          | 3            | 2002.8 |
| 4                | 1            | 4      | 4          | 4            | 2003.2 |
| 5                | 2            | 1      | 2          | 3            | 1999.7 |
| 6                | 2            | 2      | 1          | 4            | 1992.1 |
| 7                | 2            | 3      | 4          | 1            | 1998.3 |
| 8                | 2            | 4      | 3          | 2            | 1994.5 |
| 9                | 3            | 1      | 3          | 4            | 1998.4 |
| 10               | 3            | 2      | 4          | 3            | 1993.5 |
| 11               | 3            | 3      | 1          | 2            | 1995.4 |
| 12               | 3            | 4      | 2          | 1            | 1994.2 |
| 13               | 4            | 1      | 4          | 2            | 1993   |
| 14               | 4            | 2      | 3          | 1            | 1995.8 |
| 15               | 4            | 3      | 2          | 4            | 1989.7 |
| 16               | 4            | 4      | 1          | 3            | 1985.4 |

### Comparisons with existing algorithms

During the 30 independent runs of the above algorithms for 100 randomly generated examples, the minimal makespan obtained is shown in Figure 15(a) and the average makespan obtained is shown in Figure 15(b). It can be seen from Figure 15 that the optimal makespan change curve and average makespan change curve of the proposed algorithm are all located below the whole figure. Figure 16 presents the statistics of the results shown in Figure 15. In Figure 16, the first group of data is the statistics of the total number of the occasion that the minimal makespans obtained by each algorithm is the minimum among all four minimal makespan obtained by different algorithms, while the second group data are about the average makespan. In the first group of data shown in Figure 16, for the 100 instances described in Figure 15, algorithm \( ISA_PCT \) obtains the optimal solution for 100 examples, algorithm \( ISA_HGA \) obtains the optimal solution for 41 examples, algorithm \( ISA_ORMT \) obtains the optimal solution for 36 examples, and algorithm \( CSA_VCLDC \) only obtains the optimal solution for 14 examples. The

![Figure 14. The influence of the four key parameters on the performance of ISA_PCT.](image-url)
second group data of Figure 16 shows the average solution capability of each algorithm for the testing instances. Therefore, it can be seen from Figures 15 and 16 that the quality of the solution of ISA_PCT is better than that of the other three algorithms.

Figure 17 shows the average execution time of different algorithms for instances described in Figure 15. Because algorithms CSA_VCLDC and ISA_ORMT are solutions based on the standard genetic algorithm, for some instances, the execution time of algorithms CSA_VCLDC and ISA_ORMT is better than that of ISA_PCT. However, the average execution time of algorithm ISA_PCT proposed in this paper is generally better than that of algorithm ISA_HGA. In conclusion, the performance of the proposed algorithm ISA_PCT outperforms other comparison algorithms.

For the complex product shown in Figure 13, Figure 18 shows the trend of the optimal solution in each generation of the population when we apply the above algorithms to solve the instance. It can be seen from Figure 18 that the algorithm ISA_PCT obtains the optimal solution at the 33rd generation, the algorithm ISA_HGA obtains the optimal solution at the 66th generation, the algorithm ISA_ORMT obtains the optimal solution at the 78th generation, and the algorithm CSA_VCLDC only obtains the optimal at 95th
It can be seen that the convergence of the proposed algorithm is better than the other three comparison algorithms.

Conclusion and future research

For the integrated scheduling problem of complex products with tree structure, a GA-based integrated scheduling algorithm is presented. The proposed algorithm overcomes the problems of previous metaheuristic-based integrated algorithms. Based on the processing operation tree, the priority constraint table is constructed, which indirectly expresses the priority dependence relationship among operations. The designed encoding method not only guarantees the legitimacy of the initial individuals, but also solves the problem of missing solution space in the previous algorithms. The decoding method is simple but effective. The presented evolutionary operators can ensure that the operations in offspring individuals meet the priority constraints, thereby avoiding the extra detection and repair work in previous literatures. The proposed local search strategy further enhances the search ability of the algorithm for the optimal solution. The performance of the proposed algorithm is verified by the comparative experiment.

The algorithm proposed in this paper uses a random initial population method, so the quality of the initial population may be poor. Therefore, in future work, we can consider combining the designed encoding method with some heuristic rules to improve the quality of the initial population. At the same time, the evolutionary operators designed in this paper focus on ensuring the legitimacy of the offspring individuals. Therefore, it is necessary to design more effective operators to ensure that the population evolves to the most promising region. The processing machine involved in this paper is only the ordinary processing machine. In the fierce market competition environment, many enterprises have introduced flexible production lines. Therefore, the follow-up research can be carried out on the integrated scheduling problem with flexible machines. In this paper, only the single-piece tree structure complex product integrated scheduling problem is considered. In actual production applications, there are multiple products being processed at the same time, and orders may arrive at any time. Therefore, future research can consider the dynamic multi-orders integrated scheduling problem. In addition, the multi-objective complex product integrated scheduling problem is also one of the future research directions.

Declaration of conflicting interests

The author(s) declared no potential conflicts of interest with respect to the research, authorship, and/or publication of this article.

Funding

The author(s) disclosed receipt of the following financial support for the research, authorship, and/or publication of this article: We acknowledge the support of the National Natural Science Foundation of China [Grant numbers 61772160, 61370086].

ORCID iD

Zhiqiang Xie https://orcid.org/0000-0001-7422-1568

References

1. Dios M, Fernandez VV and Framinan JM. Efficient heuristics for the hybrid flow shop scheduling problem with missing operations. Comput Ind Eng 2018; 115: 88–99.
2. Gao K, Yang F, Zhou M, et al. Flexible job-shop rescheduling for new job insertion by using discrete jaya algorithm. IEEE Trans Cybern 2018; 49: 1944–1955.
3. Liu L, Liu Y and Zhang J. Learning-based hand motion capture and understanding in assembly process. IEEE Trans Ind Electron 2019; 56: 9703–9712.
4. Lee DH, Na MW, Song JB, et al. Assembly process monitoring algorithm using force data and deformation data. Rob Comput Integr Manuf 2019; 56: 149–156.
5. Rossit DA, Tohmé F and Frutos M. The non-permutation flow-shop scheduling problem: a literature review. Omega 2018; 77: 143–153.
6. Komaki G, Sheikh S and Malakooti B. Flow shop scheduling problems with assembly operations: a review and new trends. Int J Prod Res 2019; 57: 2926–2955.
7. Zhang J, Ding G, Zou Y, et al. Review of job shop scheduling research and its new perspectives under industry 4.0. J Intell Manuf 2019; 30: 1809–1830.
8. Abdullah S and Abdolrazzagh NM. Fuzzy job-shop scheduling problems: a review. *Inf Sci* 2014; 278: 380–407.

9. Xie Z, Hao S, Ye G, et al. A new algorithm for complex product flexible scheduling with constraint between jobs. *Comput Ind Eng* 2009; 57: 766–772.

10. Glover F and Greenberg HJ. New approaches for heuristic search: a bilateral linkage with artificial intelligence. *Eur J Oper Res* 1989; 39: 119–130.

11. Bayu F, Panda D, Shaik MA, et al. Scheduling of gasoline blending and distribution using graphical genetic algorithm. *Comput Chem Eng* 2020; 133: 106636.

12. Li J, Song M, Wang L, et al. Hybrid artificial bee colony algorithm for a parallel batching distributed flow-shop problem with deteriorating jobs. *IEEE Trans Cybern* 2020; 50: 2425–2439.

13. Wang Z, Zhang J and Yang S. An improved particle swarm optimization algorithm for dynamic job shop scheduling problems with random job arrivals. *Swarm Evol Comput* 2019; 51: 100594.

14. Shukla AK, Singh P and Vardhan M. An adaptive inertia weight teaching-learning-based optimization algorithm and its applications. *Appl Math Model* 2020; 77: 309–326.

15. Shao Z, Pi D and Shao W. A novel multi-objective discrete water wave optimization for solving multi-objective blocking flow-shop scheduling problem. *Knowl Based Syst* 2019; 165: 110–131.

16. Zhao F, Qin S, Zhang Y, et al. A two-stage differential biogeography-based optimization algorithm and its performance analysis. *Expert Syst Appl* 2019; 115: 329–345.

17. Zhao F, Xue F, Zhang Y, et al. A hybrid algorithm based on self-adaptive gravitational search algorithm and differential evolution. *Expert Syst Appl* 2018; 113: 515–530.

18. Khodke P and Bhongade A. Real-time scheduling in manufacturing system with machining and assembly operations: a state of art. *Int J Prod Res* 2013; 51: 4966–4978.

19. Xie Z, Liu S and Qiao P. Dynamic job-shop scheduling algorithm with dynamic set of operation having priority. *Chin J Comput* 2008; 31: 502–508.

20. Xie Z, Yang J, Zhou Y, et al. Dynamic critical paths multi-product manufacturing scheduling algorithm based on operation set. *Chin J Comput* 2011; 34: 406–412.

21. Xie Z, Gui Z and Yang J. Dynamic parallel integrated flexible scheduling algorithm based on device driver and essential path. *J Mech Eng* 2014; 50: 203–212.

22. Xie Z, Wang Y, Xin Y, et al. Integrated scheduling algorithm of device-driven mechanism between two workshops using migration time. *J Shanghai Jiaotong Univ* 2016; 50: 929–936.

23. Xie Z, Zhang X, Gao Y, et al. Time-selective integrated scheduling algorithm considering the compactness of serial processes. *J Mech Eng* 2018; 54: 191–202.

24. Wang F, Ma J, Song D, et al. Research on repair operators in the whole space search genetic algorithm of assembly job shop scheduling problem. In: 2012 7th IEEE Conference on industrial electronics and applications, Singapore, 18–20 July 2012. pp.1922–1927. New York, NY: IEEE.

25. Wang L, Jia Z and Wang F. Solving complete job shop scheduling problem using genetic algorithm. *Intell Control Autom* 2008; 51: 8307–8310.

26. Zhao S, Han Q and Wang G. Product comprehensive scheduling algorithm based on virtual component level division coding. *Comput Integr Manuf Syst* 2015; 21: 2435–2445.

27. Lei Q, Guo W and Song Y. Integrated scheduling algorithm based on an operation relationship matrix table for tree-structured products. *Int J Prod Res* 2018; 56: 5437–5456.

28. Shi F, Zhao S and Meng Y. Hybrid algorithm based on improved extended shifting bottleneck procedure and ga for assembly job shop scheduling problem. *Int J Prod Res* 2020; 58: 2604–2625.

29. Montgomery DC. *Design and analysis of experiments*. Hoboken, NJ: John Wiley & Sons, 2017.