Hindawi
Scientific Programming
Volume 2017, Article ID 2463252, 15 pages
https://doi.org/10.1155/2017/2463252

Research Article

An Extended Flexible Job Shop Scheduling Model for Flight Deck Scheduling with Priority, Parallel Operations, and Sequence Flexibility

Lianfei Yu,[1,2] Cheng Zhu,[1] Jianmai Shi,[1] and Weiming Zhang[1]

[1] *Science and Technology on Information Systems Engineering Laboratory, National University of Defense Technology, Changsha, Hunan, China*
[2] *PLA Urumqi Nationalities Commander Academy, Urumqi, Xinjiang, China*

Correspondence should be addressed to Cheng Zhu; zhucheng@nudt.edu.cn

Received 20 October 2016; Revised 24 November 2016; Accepted 29 December 2016; Published 13 February 2017

Academic Editor: Lu Zhen

Efficient scheduling for the supporting operations of aircrafts in flight deck is critical to the aircraft carrier, and even several seconds' improvement may lead to totally converse outcome of a battle. In the paper, we ameliorate the supporting operations of carrier-based aircrafts and investigate three simultaneous operation relationships during the supporting process, including precedence constraints, parallel operations, and sequence flexibility. Furthermore, multifunctional aircrafts have to take off synergistically and participate in a combat cooperatively. However, their takeoff order must be restrictively prioritized during the scheduling period accorded by certain operational regulations. To efficiently prioritize the takeoff order while minimizing the total time budget on the whole takeoff duration, we propose a novel mixed integer liner programming formulation (MILP) for the flight deck scheduling problem. Motivated by the hardness of MILP, we design an improved differential evolution algorithm combined with typical local search strategies to improve computational efficiency. We numerically compare the performance of our algorithm with the classical genetic algorithm and normal differential evolution algorithm and the results show that our algorithm obtains better scheduling schemes that can meet both the operational relations and the takeoff priority requirements.

1. Introduction

When the battle comes, carrier-based aircrafts have to take off wave by wave. The operation of the first wave of aircraft takeoff is usually the most complicated since several types of carrier-based aircrafts have to take off synergistically. Before taking off, each carrier-based aircraft requires certain supporting operations, for instance, inspection, charging, oiling, and weapon mount. Some operations can be performed simultaneously, including oiling and weapon mount. All aircrafts do not need to wait for their order since this class of operations can be processed by some different supporting groups at the same time. Nevertheless, some other operations cannot be processed at the same time. For instance, charging oxygen and oiling cannot be processed simultaneously due to the safety concern and thus aircrafts have to be processed one by one. Therefore, the latter class of supporting operations in turn affects the takeoff priority orders of aircrafts and it is important to schedule the supporting groups efficiently such that all carrier-based aircrafts take off in the minimal time horizon. The scheduling problem is usually called "deck scheduling," which can be viewed as an extension of the Flexible Job Shop Scheduling Problem (FJSP) by incorporating some practical restrictions.

Job Shop Scheduling Problem (JSP) falls in the classical group of combinatorial optimization problems and is well known to be NP-hard [1]. The Flexible Job Shop Scheduling Problem (FJSP) is an extension of the JSP [2]. However, traditional models of JSP and FJSP do not consider the typical features of the scheduling for the carrier-based aircrafts on the deck, including the following.

First, in most JSP and FJSP models, they usually restrict the precedence of operations in each job but do not consider priorities of jobs, which means any job can be completed

earlier than others. However, on the deck of the aircraft carrier, the first wave of carrier-based aircrafts may include helicopter, early warning aircraft, and fighter jets, and there are strictly takeoff orders for different carrier-based aircrafts. According to operational regulations, the helicopter must take off firstly, and then the early warning aircraft, fighter jets, which make it more difficult to schedule the supporting resources.

Second, in most JSP and FJSP models, the precedence among operations of a job is predetermined and cannot be changed. In deck scheduling, some operations of a carrier-based aircraft can be processed before or after other operations, which will increase the scheduling flexibility. For example, on the deck of the aircraft carrier, the carrier-based aircraft needs to be charged oxygen and oiled, and it is permitted to charge oxygen first, then oil, or vice versa. But for safety, the two operations cannot be processed simultaneously.

Third, the operations are processed one by one in most JSP and FJSP, but in the job of supporting carrier-based aircrafts, it will take a long time to charge oil and configure weapon sequentially. In order to save time, the two operations can be processed simultaneously.

To the best of our knowledge, only few researches considered the three typical features above in a single scheduling problem. We address these challenges in terms of the extended FJSP (EFJSP) with priority, parallel operations, and sequence flexibility. Based on FJSP, we propose a novel MILP model for the problem. Nevertheless, when the typical restrictions are incorporated into the model, few existing algorithms in the literatures can solve it efficiently. Motivated by the hardness, we develop an improved differential evolution algorithm combined with typical local search strategies. By using some practical instances flight deck scheduling and random instances, we compare the performance of the proposed algorithm with the general genetic algorithm and differential evolution algorithm. Furthermore, we also analyze the impact of the local search strategies on the performance.

2. Literature Review

FJSP has been widely studied, but most of current literatures study the scheduling problem involving one or two of the above three features, which are priority, parallel operations, and sequence flexibility. Alvarez-Valdes et al. [3] introduced some operations which can be processed simultaneously in the glass production; a heuristic algorithm combining the priority rule with local search is proposed to minimize the optimization criteria, which is based on the earliest completion and delay penalty. Vilcot and Billaut [4] discussed the process characteristics of the printing industry, where each operation has at least one former operation, but only a subsequent operation, in a complete process, and some operations can be carried out at the same time. Özgüven et al. [5] introduced a relatively simple integer programming model, in which a job can choose a feasible routing from some reverse sets. However, once the process of the job is determined, no operations can be processed simultaneously. Birgin et al. [6] proposed a MILP model for an extended version of the FJSP, which allowed the precedence to be given by an arbitrary directed acyclic graph rather than a linear order among operations of a job, and compared with [5], the result showed the accuracy of the model. In the Birgin model [6], Birgin et al. [7] put forward the scheduling list algorithm to solve the problem, further extending to the filtered beam search method. The experiment results show that the efficiency and the effect of these two methods are very good.

For the FJSP, each operation could be processed by some machines at different time. If the job has several kinds of processing routes, selecting one of them is called process planning. At first, the process planning is optimized in the case of sufficient resources, but the reality shows that this cannot fully utilize resources. Thus, Saygin and Kilic [8] established the integrated model combining process design with production scheduling, which included 5 stages: flexible process decomposition, machine tool selection, process optimization, scheduling, and rescheduling. There are some similar models such as two-level model [9], two-stage model [10], flexible process selection model [11], and mixed integer programming model [5]. However, in these studies, once the process route is selected, the order of operations is determined, and there is no essential difference with JSP and FJSP.

There are also some researches on the flight deck scheduling; Ryan et al. [12] designed an interactive local and global decision supporting system named deck operation course of action planner for aircraft carrier deck scheduling. They applied the reverse reinforcement learning method [13] and strategy optimization based on queuing network [14] in aircraft adaptive multistage scheduling problem, and through deck operation simulation, the scheduling performance of the optimization based on integer linear programming model is compared with the traditional expert heuristic rules [15]. According to the difference of carrier-based aircraft support organization of different launch mode carrier-based aircraft needed, Wei et al. [16] introduced two carrier-based aircraft support scheduling models on the basis of reasoned supposition and propriety predigesting: carrier-based aircraft support scheduling model of wavily launching carrier-based aircraft on carrier [17] and rescheduling model of carrier-based aircraft support based on mission [18]. The two models are solved based on the FJSP theory and rescheduling theory, respectively. According to the constraint of supporting process for single-carrier aircraft, a multiobjective integrated supporting scheduling model of multicarrier aircrafts was established by Han et al. [19], in which some operations can be processed simultaneously.

From the above reviews, it can be seen that most researches are concerned about one of the three features, and there are few researches which consider the three features simultaneously. Thus, the scheduling problem generated in the supporting process for carrier-based aircrafts is a new extension of the classical FJSP.

The remainder of this paper is organized as follows. In Section 3, some definitions will be proposed; the new model is

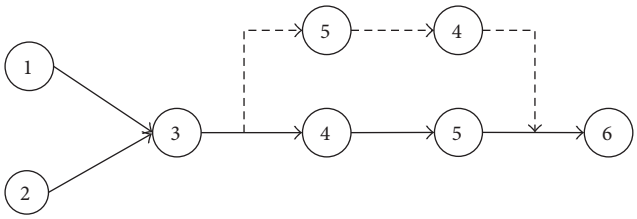

Figure 1: *The process of a simple job*. The nodes mean the operations of job, the line with arrow represents the precedence constraint between operations, and the dash line with arrow is the alternative route of job.

formalized. In Section 4, the improved DE algorithms for the extension are illustrated in detail. The computational results and analysis are provided in Section 5. Finally, we conclude the paper in Section 6.

3. The Formulation of the EFJSP Model

The deck scheduling for carrier-based aircrafts is to schedule a number of supporting groups to complete the operations for preparing the aircrafts well and let them take off as soon as possible. There are usually a set of operations to be completed before the carrier-based aircraft can take off, such as inspection, charging, oiling, and weapon mount. Each operation must be conducted by one supporting group selected from a set of candidate ones whose efficiency is different. There are also operational rules which restrict the sequence of taking off for different types of aircrafts; for instance, the helicopter must take off before all the other aircrafts. Further, the sequence of some operations can be exchanged. For example, charging oxygen is allowed to be processed before or after oiling, but not at the same time. The scheduling can be decomposed into two subproblems: the sequencing subproblem which determines a sequence for all operations and the routing subproblem that selects a suitable supporting group for each operation. The objective is to find a scheduling that minimizes the makespan.

We can see that the EFJSP is more complex than the FJSP: First, in the EFJSP, different jobs have different priority, and the final operation of the job with higher priority must be completed earlier than others with lower priority. Second, some operations of a job can be processed simultaneously, such as operations 1 and 2 in Figure 1. Third, some operations of a job can be processed by exchanging their preference order, in Figure 1, where operations 4 and 5 can be swapped.

Some definitions will be proposed based on these features which have been mentioned previously.

Definition 1 (job). All the supporting operations of a carrier-based aircraft are considered as a job.

Definition 2 (priority). When many jobs need to be done, the more important job should be completed earlier. However, for the job with higher priority, it does not mean that each operation has to be finished earlier than others; in order to make good use of the supporting groups, it is only required that the final operation must be finished earlier than others with lower priority.

Definition 3 (parallel operations). There are no precedent relations among some operations in a job, which can be processed simultaneously.

Definition 4 (sequence flexibility). This means that there are two operations in a job; they are neighbors and can change their precedent constraint but cannot be processed simultaneously.

Because the EFJSP is more complex than the FJSP, thus it is also NP-hard. In detail, the following assumptions are made in the paper:

(1) All the supporting groups are available at time 0.

(2) All the jobs can be processed at time 0.

(3) Only one operation can be processed by a supporting group at a time.

(4) Each operation must be completed without interruption once it starts.

(5) The precedence of some operations for a job is predefined and cannot be changed; some can exchange their preference orders.

(6) The setting up time of supporting groups and transferring time of operations are negligible.

The following indices, sets, and parameters are used in the new model:

i, i': indices for jobs

j, j': indices for operations

k: indices for supporting groups

n: number of jobs

m_i: number of operations in job i

M: set of supporting groups

O_{ij}: operation j of job i

M_{ij}: set of supporting groups on which operation O_{ij} can be processed, $M_{ij} \subseteq M$

P_{ij}: immediate job predecessor of operation O_{ij}

T_{ijk}: processing time if operation O_{ij} is to be performed by supporting group k

L: a very large positive number

Decision Variables

S_{ij}: starting time of operation O_{ij}

C_{ij}: completion time of operation O_{ij}

Y_{ijk}: 1 if supporting group k is assigned to operation O_{ij}; 0, otherwise

$X_{iji'j'k}$: 1 if operation O_{ij} is scheduled before operation $O_{i'j'}$ where both operations are processed by supporting group k; 0, otherwise

Z_i: 1 if operation j is processed before operation j' in job i; 0 if operation j' is processed before operation j in job i

The EFJSP is formulated into a MILP. The mathematical model is as follows:

$$\min\left(C_{\max}\right) = \min\left(\max\left(C_{im_i}\right)\right), \tag{1}$$

$$C_{ij} = S_{ij} + \sum_{k \in M_{ij}} T_{ijk} * Y_{ijk}, \tag{2}$$

$$\sum_{k \in M_{ij}} Y_{ijk} = 1, \tag{3}$$

$$\max\left(C_{ih}\right) \le S_{ij}, \quad h \in P_{ij}, \tag{4}$$

$$X_{iji'j'k} + X_{i'j'ijk} \le Y_{ijk}, \quad i \ne i', \tag{5}$$

$$X_{iji'j'k} + X_{i'j'ijk} \le Y_{i'j'k}, \quad i \ne i', \tag{6}$$

$$X_{iji'j'k} + X_{i'j'ijk} \ge 1 - \left(1 - Y_{i'j'k}\right) L - \left(1 - Y_{ijk}\right) L, \quad i \ne i', \tag{7}$$

$$S_{ij} \ge C_{i'j'} + \left(1 - \sum_{k \in M_{ij}} X_{i'j'ijk}\right) L, \quad i \ne i'. \tag{8}$$

If the priority of i is higher than i', then

$$C_{im_i} \le C_{i'm_{i'}}, \tag{9}$$

$$\left(C_{ij} - S_{ij'}\right) Z_i + \left(C_{ij'} - S_{ij}\right)\left(1 - Z_i\right) \ge 0, \quad j \ne j', \tag{10}$$

$$S_{ij} \ge 0, \tag{11}$$

$$X_{iji'j'k} \in \{0, 1\}, \tag{12}$$

$$Y_{ijk} \in \{0, 1\}, \tag{13}$$

$$Z_i \in \{0, 1\}, \tag{14}$$

$$\begin{aligned} &i, i' = 1, \ldots, n; \\ &j = 1, \ldots, m_i; \\ &j' = 1, \ldots, m_{i'}; \\ &k \in M_{ij}. \end{aligned} \tag{15}$$

The makespan is the objective function of the problem. Constraint (2) makes sure that operation is not interrupted. Constraint (3) guarantees that an operation is processed by one supporting group. Constraint (4) describes the operation precedence constraints. Constraints (5)–(8) ensure that operations cannot be processed by the same supporting group at the same time. Constraint (9) shows jobs with higher priority should be completed earlier than others with lower priority. Constraint (10) allows some operations to exchange precedence order, but the operations will not be processed simultaneously.

Comparing with the FJSP model, the EFJSP model includes more constraints, such as constraint (4), parallel operation; constraint (9), priority; and constraint (10), sequence flexibility.

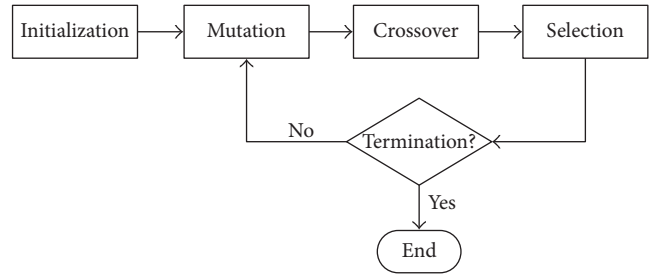

Figure 2: The flow of basic DE.

4. Proposed Improved DE for the EFJSP Model

For the FJSP, when the number of jobs is small, some exact algorithms can be employed to solve it, for example, brand-and-bound [20–23], mathematical programming [24–26], and Lagrangian relaxation [27–30]. But when the number of jobs rises, it is difficult to find an optimal solution in a short time. Many researchers have heuristic methods to solve the FJSP, such as genetic algorithm (GA) [31–34], ant colony algorithm [35], particle swarm optimization [36], tabu search [37], leapfrog algorithm [38], bee colony algorithm [39], differential evolution (DE) algorithm [40–44], and memetic algorithms [45, 46]. Some literatures study the combinations of a variety of algorithms or an algorithm with local search to obtain better performance, such as differential evolution algorithm combining with genetic algorithm [47], particle swarm optimization combining with simulated annealing [48], and tabu search combining with local search [49]. But these algorithms cannot solve the EFJSP model directly, and new improved algorithms should be explored.

DE is an optimization algorithm proposed by Storn and Price [50] for solving Chebyshev polynomials, which encodes the chromosome with floating-point vector in continuous space. DE algorithm is an intelligent optimization method based on population, which searches the entire population space with the help of the disturbance formatting by individual difference, and seeks the optimal solution of the problem through the greedy competition mechanism. The DE algorithm has less controlled parameters and is easy to understand and implement. Due to its high reliability, robustness, and good performance, DE has been widely used in optimization area. There have been some works using DE algorithm to solve FJSP and JSP [40–44], which show the advantages of the DE algorithm compared with other algorithms. The EFJSP problem is an extension of FJSP, and we made some improvement on DE by integrating some local research strategies to solve our problem more efficiently. The basic framework of DE is just as presented in Figure 2.

Table 1: Jobs and optional supporting groups.

Job	Operation	G_1	G_2	G_3
1	O_{11}	2	3	—
	O_{12}	5	6	7
	O_{13}	—	—	4
	O_{14}	6	—	8
	O_{15}	—	10	—
	O_{16}	3	5	—
2	O_{21}	5	—	7
	O_{22}	—	7	6
3	O_{31}	4	3	—
	O_{32}	7	6	5

Table 2: ID number assigned to operations.

Operation	O_{11}	O_{12}	O_{13}	O_{14}	O_{15}	O_{16}	O_{21}	O_{22}	O_{31}	O_{32}
ID	1	2	3	4	5	6	7	8	9	10

4.1. Encoding. The EFJSP model can be decomposed into two subproblems: sequencing and routing. Thus, a chromosome contains two parts; more specifically, the first part (part 1) is the sequencing that determines an operation order within a job and an operation sequence of all jobs; the second part (part 2) is the routing that assigns each operation to an appropriate supporting group. Some works [51, 52] have been done in the method of sequencing and routing.

4.1.1. Operation Sequence. The number of all operations is calculated as $d = \sum_{i=1}^{n} m_i$. Let a number (ranging from 1) denote the ID of each operation, and a smaller value of ID means a higher priority in a job.

An example of EFJSP is shown in Table 1 as an illustration, the precedent constraints of operations in job 1 are shown in Figure 1, and the precedent constraints of operations in job 2 and job 3 are liner. In Table 1, there are 3 supporting groups: G_1, G_2, and G_3. Each entry of the input table denotes the processing time of that operation on the corresponding supporting group. The tag "—" means that a supporting group cannot process the corresponding operation. The ID in Table 2 denotes the instances in Table 1.

Part 1 code can be generated by the IDs listed randomly; the length of part 1 is d. Take the instances in Table 1 as an example, and suppose there is a random operation sequence: part 1 = [8, 2, 1, 3, 5, 4, 9, 6, 7, 10], and $O_{22}(8)$ is the immediate precedent operation of $O_{21}(7)$; it is obvious that the sequence breaks the precedent constraint in job 2; as a result, part 1 code should be changed as [7, 2, 1, 3, 5, 4, 9, 6, 8, 10].

4.1.2. Supporting Group Assignment. Following the operation sequencing, each operation should select a supporting group in its alternative supporting group set. Take the instances in Table 1 as an example, and suppose part 1 is [7, 2, 1, 3, 5, 4, 9, 6, 8, 10], and number "7" denotes the operation O_{21}, which can be processed by G_1 or G_3; if the operation O_{21} is processed by G_3, then the corresponding gene is "2" in part 2. A possible supporting group assignment is shown in Table 3. A chromosome is composed of part 1 and part 2.

4.1.3. The IDs of Job with Higher Priority. In the EFJSP model, some jobs have higher priorities, which must be finished earlier; as a result, the IDs should be assigned to the jobs with higher priority first. Chromosomes which cannot satisfy the rule should be abandoned.

4.2. Decoding

4.2.1. Decoding the Chromosome to an Active Scheduling. Decoding is to convert the chromosome into a feasible solution; there are mainly three decoding methods: active scheduling, semiactive scheduling, and nondelay scheduling. Some works [53, 54] have found that the optimal scheduling to achieve the goal of makespan is an active scheduling; thus, we decode the chromosome to an active scheduling here. The steps are shown as follows.

Step 1. Set up *number* = 1.

Step 2. Take the *number*th gene in part 1 and convert it into the operation O_{ij}; then select the *number*th gene in part 2 and convert it into supporting group k; the processing time is T_{ijk} which operation O_{ij} processed by the supporting group k.

Step 3. If the operation O_{ij} is the first one to be processed by the supporting group k, $S_{ij} = \max C_{iP_{ij}}$; otherwise we should find all the idle time intervals $[TS_k, TE_k]$ in the supporting group k.

Step 4. If the length of the idle time interval is bigger than T_{ijk} according to (16), the operation O_{ij} should be inserted into the interval; otherwise, it should be allocated in the end of the supporting group k.

$$S_{ij} = \max\left\{\max C_{iP_{ij}}, TS_k\right\}, \quad S_{ij} + T_{ijk} \le TE_k. \tag{16}$$

Step 5. Repeat Steps 2 to 4 until all operations of part 1 have been processed.

4.3. Initial Population. Population initialization is crucial since it can affect the speed of the algorithm's convergence and the quality of the solution. In this part, we present a method to assign each operation to a suitable supporting group. The method focuses on the processing time and workload of supporting groups, namely, the global assignment, which is an improved one of the work [52]; the steps are as follows.

Step 1. Create a new array to record the working time of each supporting group; initial values are set as 0.

Step 2. Follow the order sequentially; take an operation from part 1.

Step 3. Add the processing time of each alternative supporting group to the corresponding location in the array; then the

Table 3: Instructions to the encoding of assigning supporting groups.

Part 1	7	2	1	3	5	4	9	6	8	10
Operation	O_{21}	O_{12}	O_{11}	O_{13}	O_{15}	O_{14}	O_{31}	O_{16}	O_{22}	O_{32}
Groups	G_1 G_3	G_1 G_2 G_3	G_1 G_2	G_3	G_2	G_1 G_3	G_1 G_2	G_1 G_2	G_2 G_3	G_1 G_2 G_3
Assigned	G_3	G_3	G_1	G_3	G_2	G_3	G_2	G_2	G_2	G_1
Part 2	2	3	1	1	1	2	2	2	1	1

corresponding supporting group with the shortest time will be selected for the operation; if there are several supporting groups with the same minimum time, the one with the least processing time should be selected.

Step 4. The order of the selected supporting group in the alternative supporting group set will be added to the corresponding gene bit of part 2.

Step 5. In the array, modify the total working time of the selected supporting group, and leave the others unchanged.

Step 6. Continue from part 1, and repeat Steps 3 to 6 until all operations are processed.

4.4. Mutation. For the mutation, DE is not suitable for discrete problems; the following two methods are applied: POX (Precedence Operation Crossover, POX) [55] and MPX (Multiple Point Crossover, MPX) [56]. The POX inherits the characteristics of operation sequence from parent chromosomes to child chromosomes; the MPX transits the features of supporting group assignment from parent chromosomes to child chromosomes.

4.4.1. POX. Precedence operation crossover is the process to cross the operation sequence in the parent chromosomes, and the supporting group assignments are reserved to the child chromosomes. The process is as follows.

First, the set of jobs $\{1, \ldots, n\}$ is divided into two random complementary and nonempty sets: S_1, S_2.

Second, all the operations of jobs in set S_1 with their assigned supporting groups are copied from "Parent 1" chromosome to "Child 1" chromosome, and all the operations of jobs in set S_1 with their assigned supporting groups are copied from "Parent 2" chromosome to "Child 2" chromosome.

Third, all the operations of jobs in set S_2 with their assigned supporting groups are copied from "Parent 2" chromosome to "Child 1" chromosome, and all the operations of jobs in set S_2 with their assigned supporting groups are copied from "Parent 1" chromosome to "Child 2" chromosome.

Taking Table 1 as an example, the process of POX is shown in Figure 3.

4.4.2. MPX. Multiple point crossover is the process to cross the supporting group assignment in the parent chromosomes, and the operation sequence is reserved to the child chromosomes. The process is as follows.

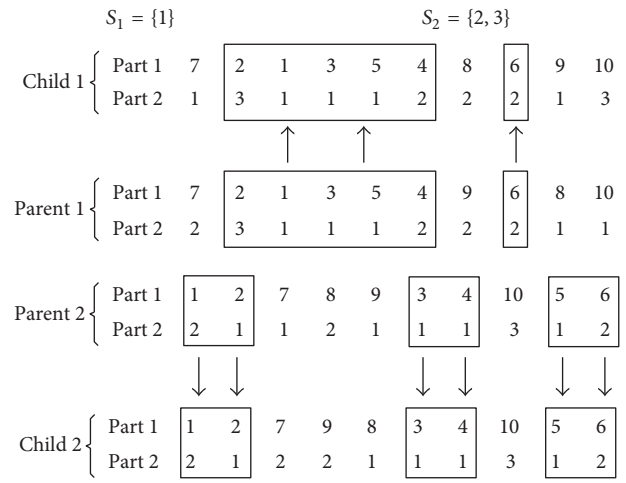

Figure 3: The process of POX.

First, an integer set R_{01} containing 0 and 1 is randomly generated, the length of which is equal to the half-length of chromosomes.

Second, the operations and their assigned supporting groups whose positions correspond to 0 in set R_{01} are copied from "Parent 1" chromosome to "Child 1" chromosome.

Third, selecting the operations in "Parent 1" chromosome whose positions correspond to 1 in set R_{01}, these operations and assigned supporting groups in "Parent 2" chromosome are copied to "Child 1" chromosome.

Similarly, for the "Child 2" chromosome, the operations and their assigned supporting groups whose positions correspond to 0 in set R_{01} are copied from "Parent 2" chromosome. Then, selecting the operations in "Parent 2" chromosome whose positions correspond to 1 in set R_{01}, these operations and assigned supporting groups in "Parent 1" chromosome are copied to "Child 2" chromosome. Taking Table 1 as an example, the process of MPX is shown in Figure 4.

The mutation of chromosome is performed by

$$V_i^g = F \otimes G\left(F \otimes G\left(X_{r_2}^g, X_{r_3}^g\right), X_{r_1}^g\right), \tag{17}$$

where $r_1 \neq r_2 \neq r_3 \neq i$, $X_{r_2}^g$ is the r_2th chromosome in the gth population, V_i^g is the ith chromosome in the gth

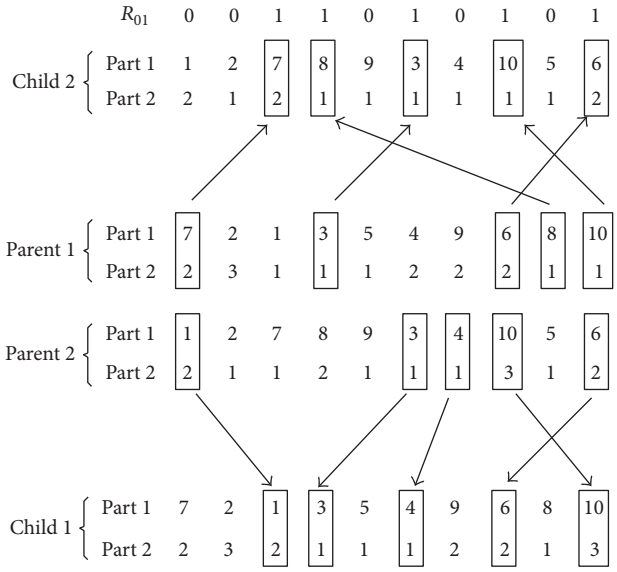

Figure 4: The process of MPX.

mutated population, F is scale factor, and $\otimes$ is cross symbol; the calculation of (17) is composed of two parts: (18) and (19).

$$Y_i^g = F \otimes G\left(X_{r_2}^g, X_{r_3}^g\right) = \begin{cases} G\left(X_{r_2}^g, X_{r_3}^g\right), & \text{rand} < F \\ X_{r_2}^g, & \text{otherwise.} \end{cases} \tag{18}$$

Equation (18) denotes the difference vector between two parent chromosomes; rand is a random number between 0 and 1. When sequencing the operations, $G(X_{r_2}^g, X_{r_3}^g)$ will be crossed in the POX way. When assigning the supporting groups, $G(X_{r_2}^g, X_{r_3}^g)$ will be crossed in the MPX way.

$$V_i^g = F \otimes G\left(Y_{r_2}^g, Y_{r_3}^g\right) = \begin{cases} G\left(Y_i^g, X_{r_1}^g\right), & \text{rand} < F \\ Y_i^g, & \text{otherwise.} \end{cases} \tag{19}$$

Equation (19) denotes the process that parent chromosomes cross with difference vector. We compute $G(Y_i^g, X_{r_1}^g)$ using the way in $G(X_{r_2}^g, X_{r_3}^g)$.

4.5. Crossover. After mutation, the crossover operator is applied to generate the trial vector U_{ij}^g as follows:

$$U_{ij}^g = \begin{cases} V_{ij}^g, & \text{rand} < \text{CR} \\ X_{ij}^g, & \text{otherwise,} \end{cases} \tag{20}$$

where $\text{CR} \in [0, 1]$ is a crossover parameter to control the diversity of the population, rand denotes a random number between 0 and 1, and the process of crossover is shown in Figure 5. When we obtain U_{ij}^g, some work needs to be done for the deduplication and completion of solution. The construction of new chromosomes should guarantee the priority of some certain operations.

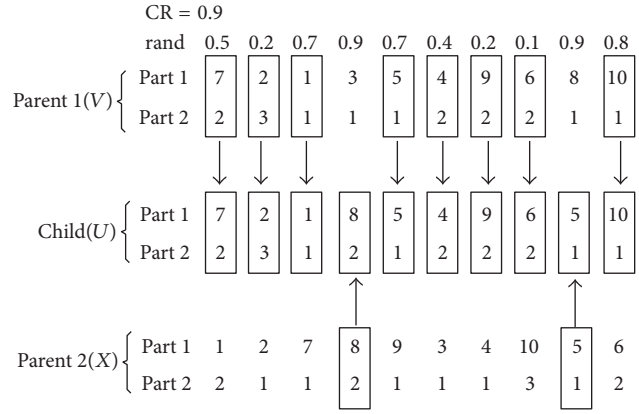

Figure 5: An illustration example of the crossover.

4.6. Selection. The selection operator is used to decide whether the trial vector U_i^g is the member of the population for the next generation, which can be described as

$$X_i^{g+1} = \begin{cases} U_i^g, & f\left(U_i^g\right) < f\left(X_i^g\right) \\ X_i^g, & \text{otherwise,} \end{cases} \tag{21}$$

where $f(\cdot)$ is the objective function to be minimized. Therefore, the trial vector U_i^g will replace the corresponding target vector X_i^g in the next generation if its objective function value is not greater than that of the target vector; otherwise, the target vector remains in the population.

4.7. Local Search Algorithm. In this part, a local search algorithm based on neighborhood structure is proposed to improve the scheduling. And the neighborhood structure is based on the critical path in the disjunctive graph, which is a kind of representation for the scheduling.

4.7.1. Disjunctive Graph. Disjunctive graph $G = (N, A, E)$ [25] is an important formulation of FJSP, where N denotes a set of all the nodes, and each node represents an operation in the EFJSP (including dummy starting and terminating operations); A represents the conjunctive arcs connecting two adjacent operations within a job, and the direction represents the processing order between the two connected operations; E denotes all disjunctive arcs connecting two adjacent operations which are processed by the same supporting group and their directions display the processing order. The processed time of each operation is generally labeled on the corresponding node and regarded as the node's weight. In Figure 6, O_{11}, O_{32} are processed by the supporting group G_1, and O_{15}, O_{31}, O_{16}, and O_{22} are executed successively by the supporting group G_2, and O_{21}, O_{12}, O_{13}, and O_{14} are performed by the supporting group G_3.

A scheduling of the EFJSP is feasible, if there are no cyclic paths in its corresponding disjunctive graph. If a disjunctive graph is acyclic, the longest path between the starting node S and the ending node E is called a critical path, whose length is defined as the makespan of the scheduling. Operations

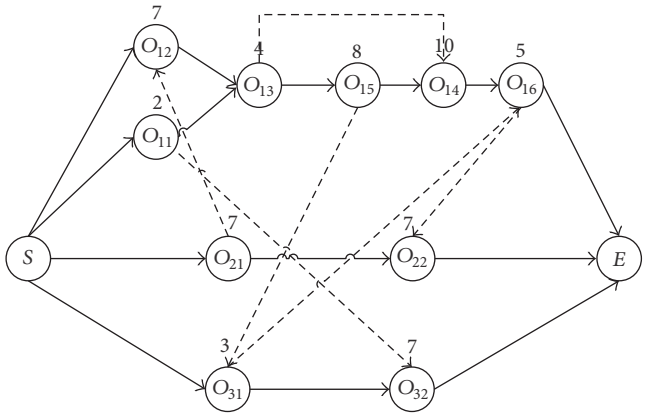

FIGURE 6: *The disjunctive graph*. The nodes mean the operations, the solid lines represent the workflow between operations in a job, and the dash lines show the workflow in a supporting group.

on the critical path are critical operations. For example, the disjunctive graph illustrated in Figure 6 is a feasible scheduling as it is acyclic, its critical path is $S \rightarrow O_{21} \rightarrow O_{12} \rightarrow O_{13} \rightarrow O_{15} \rightarrow O_{14} \rightarrow O_{16} \rightarrow O_{22} \rightarrow E$, and the makespan is 48. Here, we define $L(u, v)$ as the longest path from node u to node v.

4.7.2. Neighborhood Structures. Neighborhood structure is key to the local search, which can guide the search of algorithm effectively and avoid the blind search. The neighborhood structure of JSP is designed to make good use of the idle time in supporting groups. For the FJSP, another factor should be considered, which is the assignment of the supporting groups. For the EFJSP, the priority of jobs, parallel operations, and the flexibility of operation order will influence the neighborhood structure. Therefore, a new local search algorithm will be introduced based on the work [57]. Since the critical path defines the makespan of the scheduling, so only the critical operations should be used to search for idle time in the alternative supporting group set and implement the local search.

The following notations are used in the local search algorithm:

w: critical operation

$JP(w)$: set of immediate precedent operations of w

$JS(w)$: set of immediate succeeding operations of w

$s^E(w)$: the earliest starting time of w

$c^E(w)$: the earliest completing time of w

$s^L(w)$: the latest starting time of w

$c^L(w)$: the latest completing time of w

The idle time in the alternative supporting groups of w can be divided into two types: (1) the idle time in current supporting group which processed w and (2) the idle time in other supporting groups which are in the alternative supporting group set of w. Therefore, the move of w is also two types: (1) in current supporting group and (2) in other alternative supporting groups. Each type of idle time corresponds to a kind of local search.

The first-level local search is the moving in the current supporting group.

It is known that each operation in $JP(w)$ must be finished before the starting of w, and w must be finished before the starting of each operation in $JS(w)$. The biggest time interval of the processing w is $[\max(c^E(JP(w))), \min(s^L(JS(w)))]$.

Suppose operations x and y are neighbors, processed by the same supporting group with w. The biggest idle time interval between x and y is $[c^E(x), s^L(y)]$; if the intersection of sets $[c^E(x), s^L(y)]$ and $[\max(c^E(JP(w))), \min(s^L(JS(w)))]$ is not empty, it is possible to decrease makespan by inserting w between x and y. So the condition to move w in the current supporting group is (22) [33]:

$$\left[c^E(x), s^L(y)\right] \cap \left[\max\left(c^E(JP(w))\right), \min\left(s^L(Js(w))\right)\right] \neq 0. \tag{22}$$

The second-level local search is the moving to the other alternative supporting group.

Similarly, operations x, y are neighbors; processed by the supporting group k, which does not process w. T_{wk} is the processing time if w is processed by k. If the intersection of sets between $[c^E(x), s^L(y)]$ and $[\max(c^E(JP(w))), \min(s^L(JS(w)))]$ is bigger than T_{wk}, then it is possible to decrease the makespan by inserting w between x and y. Therefore, the condition to move w to the other supporting group k is [33]

$$\left[c^E(x), s^L(y)\right] \cap \left[\max\left(c^E(JP(w))\right), \min\left(s^L(Js(w))\right)\right] > T_{wk}. \tag{23}$$

The scheduling may not be feasible after moving the critical operations; it is important to find the conditions that can guarantee the feasibility of the scheduling when we are moving the critical operations. The work [58] proposed two rules to make sure the scheduling of FJSP is feasible when moving critical operations; we will improve these rules for the EFJSP model.

4.7.3. Rules for Moving Operation within the Current Supporting Group

Theorem 5. *For a feasible scheduling of an EFJSP model, operations u, v are processed by the same supporting group and u is precedent to v; if one of the following conditions is satisfied, then when u becomes the immediate succeeding operation of v, the scheduling is still feasible: (1) u is the final operation of job, and the set $JS(u)$ is empty; (2) $\forall u' \in JS(u)$, $L(v, E) \geq L(u', E)$, and $v \notin JS(u)$, the set $JS(u)$ is nonempty; (3) if u and v are the operations of a job, they can be swapped.*

The proof of (1) and (2) can be referred to in [58]. Under condition (3), when operations u and v are swapped, which do

Find a critical path $S \to CO_1 \to CO_2 \to \cdots \to CO_w \to E$ in a scheduling
for $i = 1$ to w **do**
  Get the alternative supporting group set CM_1 of critical operation CO_i
  for $j = 1$ to $length(CM_1)$ **do**
    Find the set $location_j$ of positions where CO_i can move into supporting group j
    for $k = 1$ to $length(location_j)$ **do**
      if j is the current supporting group processing CO_i and the condition (22) is satisfied **then**
        move operation CO_i into position k, return the new scheduling;
      end if
      if j is not the current supporting group processing CO_i and the condition (23) is satisfied **then**
        move operation CO_i into position k, return the new scheduling;
      end if
    end for
  end for
end for

Algorithm 1: The local search algorithm.

not influence other operations, the job still can be processed, so the scheduling is feasible.

Theorem 6. *For a feasible scheduling of an EFJSP model, operations u, v are processed by the same supporting group and u is precedent to v; if one of the following conditions is satisfied, then when v becomes the immediate precedent operation of u, the scheduling is still feasible: (1) v is the first operation of job, and the set $JP(v)$ is empty; (2) $\forall v' \in JP(v)$, $L(0,u) + T_u \geq L(0,v' + T_{v'})$, and $u \notin JP(v)$, the set $JP(v)$ is nonempty; (3) if u and v are the operations of a job, they can be swapped.*

The proof of (1) and (2) can be seen in [58] and the proof of (3) is the same as Theorem 5.

4.7.4. Rules for Moving Operation to the Other Supporting Groups

Theorem 7. *For a feasible scheduling of an EFJSP model, operations x, y are neighbors, processed by the same supporting group, and x is the immediate precedent operation of y; the supporting group is the alternative supporting group of w, which will be inserted between x and y; if the two following conditions are met simultaneously, then the scheduling is still feasible: (1) w is the final operation of job, and the set $JS(w)$ is empty, or $\forall w' \in JS(w)$, $L(x,E) \geq L(w',E)$, and $x \notin JS(u)$, the set $JS(w)$ is nonempty; (2) w is the first operation of job, and the set $JP(w)$ is empty, or $\forall w' \in JP(w)$, $L(0,y) + T_y \geq L(0,w' + T_{w'})$, and $y \notin JP(w)$, the set $JP(w)$ is nonempty.*

The proof can be referred to in [58].

4.7.5. Local Search Algorithm. With the three theorems, the scheduling obtained after moving the critical operations is feasible. The detailed procedure to obtain acceptable scheduling from the neighborhood structure is described in Algorithm 1.

4.8. Improved DE with Local Search. The improved DE with local search is depicted as follows.

Step 1. Set the size of the population NP, scale factor F, crossover probability CR, and maximum number of generations $G_{\max}$.

Step 2. Initialize the population, when assigning supporting groups, the number of chromosomes selecting the supporting group is $0.8 * \text{NP}$ with the method of the global assignment, and the others select their supporting groups randomly.

Step 3. Evaluate each chromosome.

Step 4 (mutation). Generate NP donor vectors V_i^g, $i = 1, \ldots, \text{NP}$.

Step 5 (crossover). Generate NP trial vectors U_i^g, $i = 1, \ldots, \text{NP}$.

Step 6 (selection). Determine NP target vectors X_i^{g+1}, $i = 1, \ldots, \text{NP}$, by one-to-one selection operator for the next generation.

Step 7 (local search). Choose the chromosome whose makespan is the biggest one in X_i^{g+1}, $i = 1, \ldots, \text{NP}$, perform the local search to the chromosome, and if a new chromosome is found whose makespan is smaller than that one, then it will be replaced by the new chromosome.

Step 8. If $G < G_{\max}$, then repeat Steps 4 to 7; otherwise, stop.

5. Experiment Design and Computational Results Analysis

In order to analyze the performance of the improved differential evolution algorithm with local search strategies (IDE&LS) proposed in Section 4, we test the algorithm through some practical instances from flight deck scheduling. All the instances are also solved by the general genetic algorithm (GA), differential evolution (DE) algorithm, and improved differential evolution (IDE) algorithm without local search

Table 4: Supporting group and its supporting operation and supporting time.

Supporting group	(Operation, time)	Supporting group	(Operation, time)
1	(A, 2)	16	(G, 2)
2	(A, 3)	17	(H, 1)
3	(B, 2)	18	(H, 2)
4	(B, 2)	19	(I, 13)
5	(C, 3)	20	(I, 14)
6	(C, 2)	21	(I, 15)
7	(D, 4)	22	(I, 16)
8	(D, 5)	23	(J, 3)
9	(E, 11)	24	(J, 4)
10	(E, 12)	25	(K, 6)
11	(E, 10)	26	(K, 7)
12	(E, 11)	27	(L, 2)
13	(F, 7)	28	(L, 3)
14	(F, 8)	29	(L, 4)
15	(G, 4)	30	(L, 5)

strategies; their computational results are compared with that of the proposed algorithm. In the encodings of GA and DE, the supporting group is selected randomly, and the chromosome is not decoded to an active scheduling.

5.1. Experiment Design. All the methods are implemented in MATLAB 2013(a) on 2.50 GHz Intel Core i5-3210m CPU, 4 GB of RAM, Windows 7.0. There are five instances based on the flight deck scheduling. In each instance, there are 5 types of carrier-based aircrafts considered, which include a helicopter (number 1), a warning airplane (number 2), an escort fighter (number 3), an antisubmarine aircraft (number 4), and some attack fighters. In different instances, the amounts of attack fighters are different.

Instance 1. There are four attack fighters (numbers 5–8).

Instance 2. There are six attack fighters (numbers 5–10).

Instance 3. There are eight attack fighters (numbers 5–12).

Instance 4. There are ten attack fighters (numbers 5–14).

Instance 5. There are twelve attack fighters (numbers 5–16).

The order of taking off is helicopter, warning airplane, escort fighter, antisubmarine aircraft, and attack fighters. The supporting operations of a carrier-based aircraft are shown in Figure 7; there are 12 operations (A–L, numbers 1–12) to support a carrier-based aircraft, but some carrier-based aircrafts are different; for example, the helicopter and warning airplane are without weapon mount, and the helicopter does not need to catapult.

There are 30 supporting groups; operations that each supporting group can process and processing time (in minutes) are shown in Table 4.

Different population sizes and iterations are tested, and computational results showed that all algorithms converge fast when population size NP = 100, and iterations $G_{\max}$ = 400. For other parameters of the algorithms, [59] presents some experiential value for the crossover probability, mutation probability, and GGAP for the GA, and [60] suggests the value of scale factor and crossover probability in most cases for the DE. Based on their studies, we conducted some sensitivity analysis on the value of the parameters, which help us find their suitable value in our problem.

Figure 8 presents the computational results for the IDE&LS to solve Instance 3, when the value of scale factor F varies from 0.1 to 0.9. The makespan is the average value for results of 100 times' run of the algorithm. It can be seen that the best solution is obtained when $F = 0.5$. Figure 9 presents the results when the value of CR changes from 0.1 to 0.95, and we can see that the best solution is obtained at CR = 0.9. Similarly, we estimate the value for other parameters. Thus, for DE, IDE, and IDE&LS, we set NP = 100, $F = 0.5$, CR = 0.9, and $G_{\max} = 400$, while for GA, we set NP = 100, $px = 0.8$, $pm = 0.06$, GGAP = 0.9, and $G_{\max} = 400$.

5.2. Experimental Results and Analysis. Each method is run 100 times in each instance; calculating the average results, the experiment results are shown in Table 5, and the "makespan" is the object (in minutes).

Table 5 summaries the average performance of four methods in five experimental instances. The best results are presented in bold. The statistically significant improvements of the best result over competitors are indicated with asterisks $*$ (two-sample t-test at the default 0.05 significance level), followed by p value in brackets [61]. As we can see from Table 5, IDE&LS always get the lowest average value and statistically better than GA and DE in all settings. Compared with the state-of-the-art IDE, IDE&LS (81.10) still have statistically lower value than IDE (83.40) in the fifth case. These results demonstrate the benefit of the proposed method in this paper.

Table 5: The average makespan of each method and relevant p value.

Instance	GA	DE	IDE	IDE&LS
1	$52.75^{*(0.00)}$	$51.90^{*(0.01)}$	51.15	**50.95**
2	$62.05^{*(0.00)}$	$61.95^{*(0.00)}$	59.25	**58.80**
3	$72.05^{*(0.00)}$	$70.55^{*(0.00)}$	67.40	**66.50**
4	$81.45^{*(0.00)}$	$79.30^{*(0.00)}$	75.70	**74.85**
5	$92.15^{*(0.00)}$	$87.55^{*(0.00)}$	$83.40^{*(0.00)}$	**81.10**

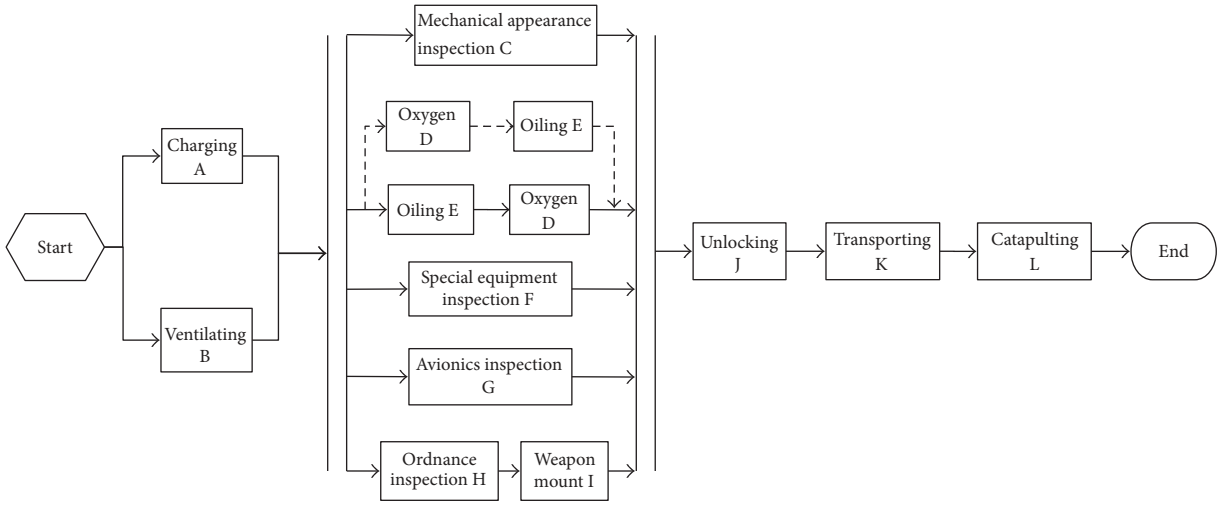

Figure 7: The supporting process of a carrier-based aircraft.

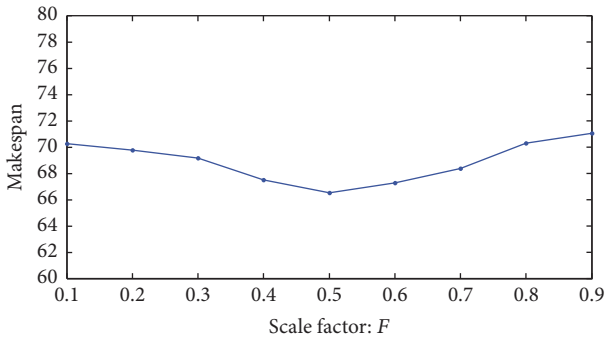

Figure 8: The average makespan when CR = 0.9; F changes from 0.1 to 0.9.

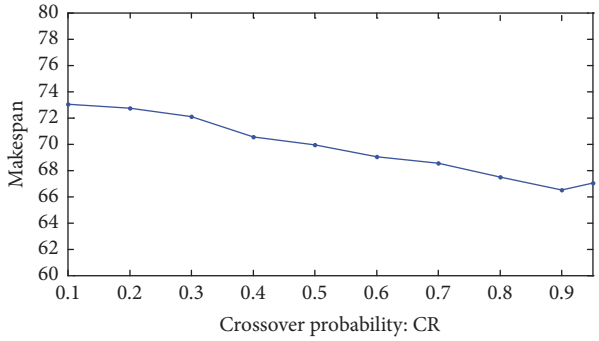

Figure 9: The average makespan when F = 0.5; CR changes from 0.1 to 0.95.

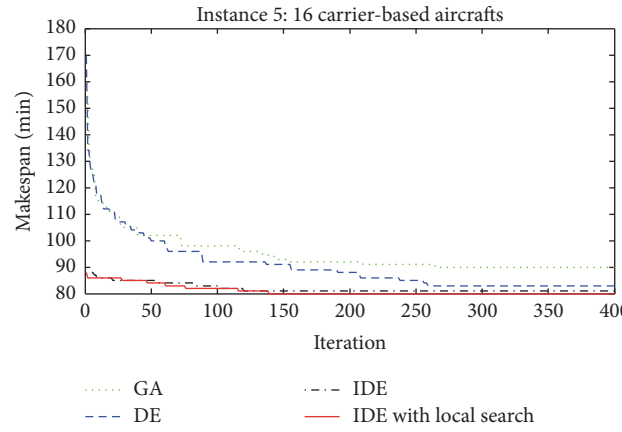

Figure 10: The convergence curves of the four methods in Instance 5.

For Instance 5, one experimental result of each method is selected randomly from 100 and its convergence curve is shown in Figure 10.

From the above computational results, we can see that the IDE&LS can obtain much better solutions than GA, DE, and IDE without local search, and also it converges much faster than other methods in the same number of iterations. That is because in the initializing process of the IDE&LS, we utilized some heuristic strategies to generate better and feasible chromosomes, which increase the search efficiency

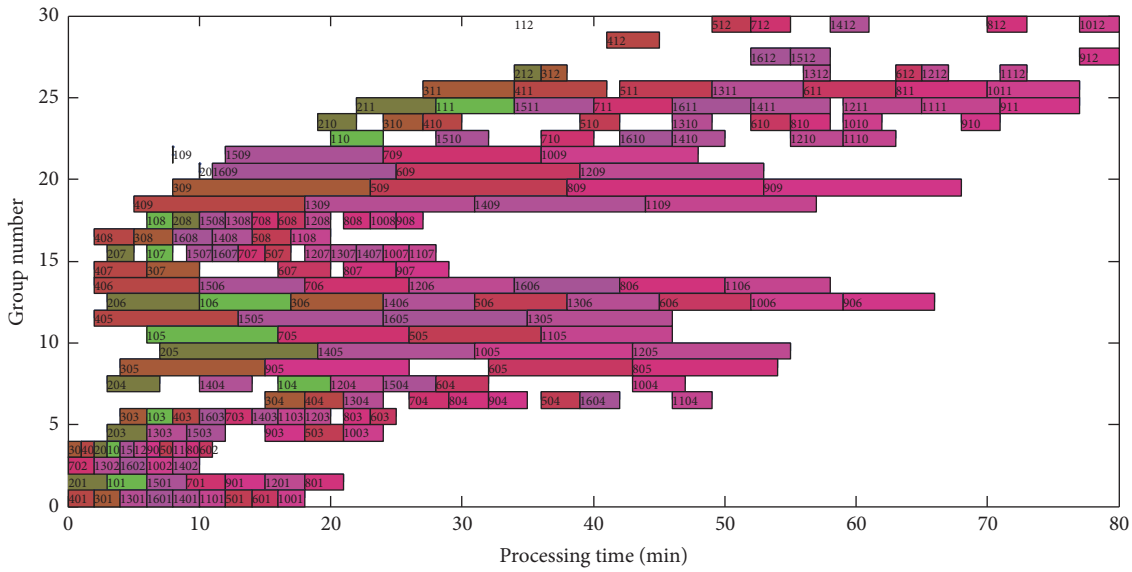

Figure 11: The scheduling of Instance 5 by IDE&LS.

in the beginning of the algorithm. Further, the local search strategies integrated in the method help to improve the quality of chromosomes of each generation, which increase the search quality.

In the military operation, the scheduling plans of supporting groups for different situations are usually made before the beginning of the battle, and the commanders concern more on the optimality of the plan. Thus, in the above analysis, we compared the average makespan of the algorithms while ignoring the computing time. Our computational experiments show that, even for the largest scale instance, the computing time of IDE&LS is not over 11 minutes, which is reasonable and can satisfy the requirement of military operation.

5.3. A Feasible Scheduling Scheme. In order to check the priority, parallel operation, and sequence flexibility, a scheduling Gantt chart of Instance 5 is generated by IDE&LS; as shown in Figure 11, the vertical axis represents the number of the supporting group, the horizontal axis represents the processing time, the first or first two digits of the digital representation are the number of carrier-based aircraft, and the latter two represent the number of operation: for example, "507" indicates operation 7 (operation G) of the carrier-based aircraft 5 and "1203" indicates operation 3 (operation C) of the carrier-based aircraft 12.

The results in Figure 11 show the following:

(1) There are some vertical lines, which mean the carrier-based aircrafts have no operations: for example, "112" indicates that number 1 carrier-based aircraft has no operation 12 (operation G), which is in line with the reality: the helicopter does not need catapult to take off.

(2) Departure time (operation G) is sorted according to helicopter (number 1), warning airplane (number 2), escort fighter (number 3), antisubmarine aircraft (number 4), and fighters (numbers 5–12), because fighters have same priorities, so anyone of which is allowed to take off firstly.

(3) In the maintenance of each carrier-based aircraft, there are some operations being processed simultaneously.

(4) For carrier-based aircrafts with higher priorities, not all the operations of them must be finished earlier than those with lower priorities, as long as the last operation can be completed earlier. For example, the helicopter (number 1) should take off first, then warning airplane (number 2), but in Figure 11, operation A of warning airplane is processed earlier than helicopter.

(5) Operation 4 (operation D) is processed before operation 5 (operation E) for some carrier-based aircrafts, such as numbers 2, 6, 8, 12, 13, and 14; the others are opposite. In conclusion, the scheduling is feasible.

6. Conclusions

In this paper, we investigated a complex scheduling problem drawn from the deck scheduling of supporting operations for carrier-based aircrafts, where precedence constraints, parallel operations, and sequence flexibility should be considered. Such scheduling problem also commonly appears in production and maintenance of huge items, like aircraft, ship, and its engine. An EFJSP model is established to formulate the scheduling problem, and an improved differential evolution algorithm was proposed to solve the problem. Firstly, the codes were initialized with the shortest time and load balancing strategy, in which the random generation and the priority strategies are also integrated. Secondly, the decoding was done in the way of activity scheduling,

reflecting the parallel, flexible features of the operations, and the combination of POX cross and MOX cross was used to make chromosomes mutation. Instances based on practical deck scheduling are utilized to test the proposed model and algorithm. Computational results show that our solution approach can present better solutions in reasonable computational time.

Besides the typical flight deck scheduling problem, the approach developed in this paper can also be extended to other classes of huge item production and maintenance scheduling problems, such as aircraft, ship, and its engine.

The problem studied in the paper is a new one, and we solve it by improving the differential evolution algorithm. However, many other algorithms may be improved and used to solve the deck scheduling problem, such as ant colony algorithm, particle swarm optimization, leapfrog algorithm, and bee colony algorithm. Thus, it is an important extension to work on the improvement of these algorithms to make them able to solve the proposed deck scheduling problem and compare with the IDE algorithm in this paper, which may help find more efficient algorithms in future.

Competing Interests

The authors declare that they have no competing interests.

Acknowledgments

This research was supported by the National Natural Science Foundation no. 61273322.

References

[1] M. R. Garey, D. S. Johnson, and R. Sethi, "The complexity of flowshop and jobshop scheduling," *Mathematics of Operations Research*, vol. 1, no. 2, pp. 117–129, 1976.

[2] P. Brucker and R. Schlie, "Job-shop scheduling with multi-purpose machines," *Computing*, vol. 45, no. 4, pp. 369–375, 1990.

[3] R. Alvarez-Valdes, A. Fuertes, J. M. Tamarit, G. Giménez, and R. Ramos, "A heuristic to schedule flexible job-shop in a glass factory," *European Journal of Operational Research*, vol. 165, no. 2, pp. 525–534, 2005.

[4] G. Vilcot and J.-C. Billaut, "A tabu search and a genetic algorithm for solving a bicriteria general job shop scheduling problem," *European Journal of Operational Research*, vol. 190, no. 2, pp. 398–411, 2008.

[5] C. Özgüven, L. Özbakır, and Y. Yavuz, "Mathematical models for job-shop scheduling problems with routing and process plan flexibility," *Applied Mathematical Modelling*, vol. 34, no. 6, pp. 1539–1548, 2010.

[6] E. G. Birgin, P. Feofiloff, C. G. Fernandes, E. L. de Melo, M. T. Oshiro, and D. P. Ronconi, "A MILP model for an extended version of the flexible job shop problem," *Optimization Letters*, vol. 8, no. 4, pp. 1417–1431, 2014.

[7] E. G. Birgin, J. E. Ferreira, and D. P. Ronconi, "List scheduling and beam search methods for the flexible job shop scheduling problem with sequencing flexibility," *European Journal of Operational Research*, vol. 247, no. 2, pp. 421–440, 2015.

[8] C. Saygin and S. E. Kilic, "Integrating flexible process plans with scheduling in flexible manufacturing systems," *The International Journal of Advanced Manufacturing Technology*, vol. 15, no. 4, pp. 268–280, 1999.

[9] Y. F. Zhang, A. N. Saravanan, and J. Y. H. Fuh, "Integration of process planning and scheduling by exploring the flexibility of process planning," *International Journal of Production Research*, vol. 41, no. 3, pp. 611–628, 2003.

[10] A. Jain, P. K. Jain, and I. P. Singh, "An integrated scheme for process planning and scheduling in FMS," *The International Journal of Advanced Manufacturing Technology*, vol. 30, no. 11-12, pp. 1111–1118, 2006.

[11] X. Y. Li, X. Y. Shao, and L. Gao, "Optimization of flexible process planning by genetic programming," *The International Journal of Advanced Manufacturing Technology*, vol. 38, no. 1-2, pp. 143–153, 2008.

[12] J. C. Ryan, M. L. Cummings, N. Roy, A. Banerjee, and A. Schulte, "Designing an interactive local and global decision support system for aircraft carrier deck scheduling," in *Proceedings of the AIAA Infotech at Aerospace Conference and Exhibit*, St. Louis, Miss, USA, March 2011.

[13] B. Michini and J. P. How, "A human-interactive course of action planner for aircraft carrier deck operations," in *Proceedings of the AIAA Infotech at Aerospace Conference and Exhibit*, St. Louis, Miss, USA, March 2011.

[14] R. G. Dastidar and E. Frazzoli, "A queueing network based approach to distributed aircraft carrier deck scheduling," in *Proceedings of the AIAA Infotech at Aerospace Conference and Exhibit*, St. Louis, Miss, USA, March 2011.

[15] J. C. Ryan, A. G. Banerjee, M. L. Cummings, and N. Roy, "Comparing the performance of expert user heuristics and an integer linear program in aircraft carrier deck operations," *IEEE Transactions on Cybernetics*, vol. 44, no. 6, pp. 761–773, 2014.

[16] C. Q. Wei, C. l. Chen, and Br. Wang, "Research on the aircraft support scheduling model of carrier-based aircraft based on launch mode," *Journal of Naval Aeronautical and Astronautical University*, vol. 27, no. 1, pp. 111–114, 2012.

[17] C. Wei, C. Chen, and B. Wang, "Study of aircraft support scheduling of waverly launching aircraft on carrier," *Control Engineering of China*, vol. 19, pp. 108–111, 2012.

[18] C. Q. Wei, C. l. Chen, and Br. Wang, "Rescheduling study of aircraft support of running launch aircraft on carrier based on mission," *Command Control & Simulation*, vol. 34, no. 3, pp. 23–27, 2012.

[19] W. Han, X. Su, and J. Chen, "Research on integrated maintenance support scheduling method of multi-carrier aircrafts," *Systems Engineering and Electronics*, vol. 35, no. 2, pp. 338–344, 2014.

[20] S. Ashour and S. R. Hiremath, "A branch-and-bound approach to the job-shop scheduling problem," *International Journal of Production Research*, vol. 11, no. 1, pp. 47–58, 1973.

[21] P. Brucker, B. Jurisch, and B. Sievers, "A branch and bound algorithm for the job-shop scheduling problem," *Discrete Applied Mathematics*, vol. 49, no. 1–3, pp. 107–127, 1994.

[22] S. Fernandes and H. R. Lourenço, "A GRASP and branch-and-bound metaheuristic for the job-shop scheduling," in *Evolutionary Computation in Combinatorial Optimization: 7th European Conference, EvoCOP 2007, Valencia, Spain, April 11–13, 2007. Proceedings*, vol. 4446 of *Lecture Notes in Computer Science*, pp. 60–71, Springer, Berlin, Germany, 2007.

[23] N. Melab, I. Chakroun, M. Mezmaz, and D. Tuyttens, "A GPU-accelerated branch-and-bound algorithm for the flow-shop

scheduling problem," in *Proceedings of the IEEE International Conference on Cluster Computing (CLUSTER '12)*, pp. 10–17, Beijing, China, September 2012.

[24] A. S. Manne, "On the job-shop scheduling problem," *Operations Research*, vol. 8, pp. 219–223, 1960.

[25] E. Balas, "Machine sequencing via disjunctive graphs: an implicit enumeration algorithm," *Operations Research*, vol. 17, no. 6, pp. 941–957, 1969.

[26] M. Van Hulle, "A goal programming network for mixed integer linear programming: a case study for the job-shop scheduling problem," *International Journal of Neural Systems*, vol. 2, no. 3, pp. 201–209, 1991.

[27] M. L. Fisher, "Optimal solution of scheduling problems using lagrange multipliers: part I," *Operations Research*, vol. 21, no. 5, pp. 1114–1127, 1973.

[28] H. Chen, C. Chu, and J.-M. Proth, "More efficient Lagrangian relaxation approach to job-shop scheduling problems," in *Proceedings of the IEEE International Conference on Robotics and Automation*, vol. 1, pp. 496–501, Aichi, Japan, May 1995.

[29] P. B. Luh, X. Zhao, Y. Wang, and L. S. Thakur, "Lagrangian relaxation neural networks for job shop scheduling," *IEEE Transactions on Robotics and Automation*, vol. 16, no. 1, pp. 78–88, 2000.

[30] R. Buil, M. À. Piera, and P. B. Luh, "Improvement of Lagrangian relaxation convergence for production scheduling," *IEEE Transactions on Automation Science and Engineering*, vol. 9, no. 1, pp. 137–147, 2012.

[31] I. Kacem, S. Hammadi, and P. Borne, "Approach by localization and multiobjective evolutionary optimization for flexible job-shop scheduling problems," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 32, no. 1, pp. 1–13, 2002.

[32] H. Z. Jia, A. Y. C. Nee, J. Y. H. Fuh, and Y. F. Zhang, "A modified genetic algorithm for distributed scheduling problems," *Journal of Intelligent Manufacturing*, vol. 14, no. 3-4, pp. 351–362, 2003.

[33] J. Gao, L. Sun, and M. Gen, "A hybrid genetic and variable neighborhood descent algorithm for flexible job shop scheduling problems," *Computers and Operations Research*, vol. 35, no. 9, pp. 2892–2907, 2008.

[34] F. Pezzella, G. Morganti, and G. Ciaschetti, "A genetic algorithm for the flexible job-shop scheduling problem," *Computers & Operations Research*, vol. 35, no. 10, pp. 3202–3212, 2008.

[35] L. N. Xing, Y. W. Chen, P. Wang, Q. S. Zhao, and J. Xiong, "A knowledge-based ant colony optimization for flexible job shop scheduling problems," *Applied Soft Computing*, vol. 10, no. 3, pp. 888–896, 2010.

[36] B. S. Girish and N. Jawahar, "A particle swarm optimization algorithm for flexible job shop scheduling problem," in *Proceedings of the IEEE International Conference on Automation Science and Engineering (CASE '09)*, pp. 298–303, Bangalore, India, August 2009.

[37] M. Saidi-Mehrabad and P. Fattahi, "Flexible job shop scheduling with tabu search algorithms," *The International Journal of Advanced Manufacturing Technology*, vol. 32, no. 5-6, pp. 563–570, 2007.

[38] K. Lu, L. Ting, W. Keming, Z. Hanbing, T. Makoto, and Y. Bin, "An improved shuffled frog-leaping algorithm for flexible job shop scheduling problem," *Algorithms (Basel)*, vol. 8, no. 1, pp. 19–31, 2015.

[39] J.-Q. Li, Q.-K. Pan, and M. F. Tasgetiren, "A discrete artificial bee colony algorithm for the multi-objective flexible job-shop scheduling problem with maintenance activities," *Applied Mathematical Modelling*, vol. 38, no. 3, pp. 1111–1132, 2014.

[40] Q.-K. Pan, L. Wang, and B. Qian, "A novel differential evolution algorithm for bi-criteria no-wait flow shop scheduling problems," *Computers and Operations Research*, vol. 36, no. 8, pp. 2498–2511, 2009.

[41] Q.-K. Pan, L. Wang, L. Gao, and W. D. Li, "An effective hybrid discrete differential evolution algorithm for the flow shop scheduling with intermediate buffers," *Information Sciences*, vol. 181, no. 3, pp. 668–685, 2011.

[42] L. Tang, Y. Zhao, and J. Liu, "An improved differential evolution algorithm for practical dynamic scheduling in steelmaking-continuous casting production," *IEEE Transactions on Evolutionary Computation*, vol. 18, no. 2, pp. 209–225, 2014.

[43] G. Deng and X. Gu, "A hybrid discrete differential evolution algorithm for the no-idle permutation flow shop scheduling problem with makespan criterion," *Computers and Operations Research*, vol. 39, no. 9, pp. 2152–2160, 2012.

[44] Y. Yuan and H. Xu, "Flexible job shop scheduling using hybrid differential evolution algorithms," *Computers & Industrial Engineering*, vol. 65, no. 2, pp. 246–260, 2013.

[45] T. C. Chiang and H. J. Lin, "Flexible job shop scheduling using a multi-objective memetic algorithm," in *Proceedings of the International Conference on Intelligent Computing*, pp. 49–56, Springer, Berlin, Germany, 2011.

[46] Y. Yuan and H. Xu, "Multiobjective flexible job shop scheduling using memetic algorithms," *IEEE Transactions on Automation Science and Engineering*, vol. 12, no. 1, pp. 336–353, 2015.

[47] Y.-W. Zhao, H.-Y. Wang, X.-L. Xu, and W.-L. Wang, "A new hybrid parallel algorithm for consistent-sized batch splitting job shop scheduling on alternative machines with forbidden intervals," *The International Journal of Advanced Manufacturing Technology*, vol. 48, no. 9–12, pp. 1091–1105, 2010.

[48] W. Xia and Z. Wu, "An effective hybrid optimization approach for multi-objective flexible job-shop scheduling problems," *Computers & Industrial Engineering*, vol. 48, no. 2, pp. 409–425, 2005.

[49] N. Liouane, I. Saad, S. Hammadi, and P. Borne, "Ant systems & local search optimization for flexible job shop scheduling production," *International Journal of Computers Communications & Control*, vol. 2, no. 2, pp. 174–184, 2014.

[50] R. Storn and K. Price, "Differential evolution—a simple and efficient adaptive scheme for global optimization over continuous spaces," Tech. Rep. TR-95–012, International Computer Science Institute, Berkeley, Calif, USA, 1995.

[51] N. B. Ho, J. C. Tay, and E. M.-K. Lai, "An effective architecture for learning and evolving flexible job-shop schedules," *European Journal of Operational Research*, vol. 179, no. 2, pp. 316–333, 2007.

[52] H. Liu, A. Abraham, and C. Grosan, "A novel variable neighborhood particle swarm optimization for multi-objective flexible job-shop scheduling problems," in *Proceedings of the 2nd International Conference on Digital Information Management (ICDIM '07)*, vol. 2, Lyon, France, October 2007.

[53] Z. Guohui, *Research on Methods for Flexible Job Shop Scheduling Problems*, Huazhong University of Science and Technology, Wuhan, China, 2009.

[54] M. Pindeo, *Scheduling: Theory, Algorithms, and Systems*, Prentice Hall, New York, NY, USA, 2002.

[55] C. Zhang, Y. Rao, and X. Liu, "An improved genetic algorithm for the job-shop scheduling problem," *China Mechanical Engineer*, vol. 15, no. 23, pp. 2149–2153, 2004.

[56] G. Shi, "A genetic algorithm applied to a classic job-shop scheduling problem," *International Journal of Systems Science*, vol. 28, no. 1, pp. 25–32, 1997.

[57] S. Zhao, "Bilevel neighborhood search hybrid algorithm for the flexible job shop scheduling problem," *Journal of Mechanical Engineering*, vol. 51, no. 14, pp. 175–184, 2015.

[58] S.-K. Zhao, S.-L. Fang, and X.-J. Gu, "Idle time neighborhood search genetic algorithm for job shop scheduling," *Computer Integrated Manufacturing Systems*, vol. 20, no. 8, pp. 1930–1940, 2014.

[59] F. Shi, H. Wang, L. Yu, and F. Hu, *MATLAB Intelligent Algorithm 30 Cases Analysis*, Beihang University Press, 2014.

[60] C. Zhang, *Theory and Application: Differential Evolution Algorithm*, Beijing Institute of Technology Press, 2014.

[61] J. Derrac, S. García, D. Molina, and F. Herrera, "A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms," *Swarm and Evolutionary Computation*, vol. 1, no. 1, pp. 3–18, 2011.