Received May 11, 2021, accepted July 6, 2021, date of publication July 9, 2021, date of current version August 2, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3096007

A Hybrid Genetic Algorithm for Flexible Job Shop Scheduling Problem With Sequence-Dependent Setup Times and Job Lag Times

YILUN WANG AND QIANWEN ZHU
School of Management and Engineering, Nanjing University, Nanjing 210093, China
Corresponding author: Yilun Wang (yilun@smail.nju.edu.cn)

ABSTRACT This paper addresses the flexible job shop scheduling problem with sequence-dependent set-up times and job lag times (FJSP-SDST-LT), which characteristics are important in modern manufacturing systems. We first present a mathematical model with the objective to minimize the makespan. Then a hybrid algorithm (HGA-TS) which combines genetic algorithm (GA) and tabu search (TS) is proposed to solve the FJSP-SDST-LT. The GA performs powerful global search by genetic operators and serves as exploration, while based on the specific structure of SDST and LT, the TS is able to perform an effective local search by relocating operations and serves as exploitation. Therefore, the proposed HGA-TS integrates good searching ability with strong diversifying ability. In order to solve the FJSP-SDST-LT effectively, we adopt effective encoding and decoding methods, genetic operators in GA, and four neighborhood structures in TS. We conduct computational experiments on two classes of instances generated from two classic data sets and the results show the great performance of HGA-TS in solving FJSP-SDST-LT.

INDEX TERMS Flexible job shop scheduling, genetic algorithm, hybrid algorithm, job lag times, sequence-dependent setup times, tabu search.

I. INTRODUCTION

Production scheduling is one of the most critical optimization issues in the planning and managing of the modern manufacturing systems [1]. The flexible job shop scheduling problem(FJSP) was first presented by [2], which is one of the most prevalent problems, because it is widely applicable to real-world manufacturing systems. Since then extensive research has been carried out by researchers [3] and [4]. However in modern manufacturing systems with many practical scheduling settings, FJSP is no longer suitable with the assumption that setup times are zero or constant and the lag times are neglected after operation completion. In many real applications such as chemical, semiconductor, pharmaceutical and automobile industry, setup time is incurred when two operations are successively processed in the same machine and lag time is incurred when an operation is finished and demands time of cooling down before the next procedure [5]. In this case, the flexible job shop problem with sequence-dependent setup times and lag times (FJSP-SDST-LT) is extended from the FJSP.

Research on extensions of FJSP is abundant. The FJSP with setup times has been studied for decades. A recent survey [3] considered scheduling problems with sequence-dependent setup times on parallel machines. Reference [6] presented a hierarchical tabu search algorithm to solve the FJSP with setup times. The algorithm was composed of a procedure that searched for the best sequence of job operations, and a procedure that found the best choice of machine alternatives. Reference [7] solved the FJSP with attached and detached setup times, machine release dates, and technological lag times constraints by a particle swarm optimization-based algorithm. Based on the previous study, [8] presented a two-stage genetic algorithm for this problem, where a new encoding method and two searching stages were introduced. Reference [9] solved a realistic manufacturing problem with transportation time and sequence-dependent setup times by an ant colony optimization method. Later, [10] proposed a swarm intelligence approach to solve FJSP with setup times, transportation times, and various release dates of each job, where the

The associate editor coordinating the review of this manuscript and approving it for publication was Christian Pilato.

assignment and sequencing problem were solved in a single step. Reference [11] developed an extensive neighborhood search function from [12] for FJSP with setup times and assumed a definition of setup times which was symmetric and satisfied the triangular inequality property. Reference [13] presented an MILP of FJSP-SDST which solved small instances to optimality and developed a tabu search algorithm with specific neighborhood structures. Reference [14] presented an exact and heuristic decomposition-based solution approaches to solve the FJSP-SDST with the objective to minimize the makespan and the total tardiness.

Other practical settings were also incorporated into the basic FJSP. In some practical settings, operations may not be independent and they may interrelate with other operations because of assembly requirements, while in some cases finished operations may not be instantly ready for the next procedure because of the necessity for drying, cooling or other ancillary operations. Reference [7] and [15] formulated an MILP for the complex FJSP in assembly job shop systems. Reference [5] discussed real-world applications of scheduling problems where steel needs to be cooled to form slabs. Reference [16] and [17] solved the issue of lot streaming in FJSP, where batches of the jobs were split into subplots to allow the overlapping of operations.

Although plenty research has been conducted on the extension of FJSP, additional features combining SDST and LT have not been studied adequately. Considering its complexity status and broad application, an effective algorithm for the FJSP-SDST-LT is desirable. GA is an effective meta-heuristics which shows great performance for scheduling problems on account of its powerful global search ability. Reference [18] proposed a GA that integrated different strategies for generating the initial population, selecting the individuals for reproduction, and reproducing new individuals. Reference [19] proposed an improved GA to solve the distributed FJSP. Reference [20] developed the algorithm based on GA and Grouping Genetic Algorithm (GGA) for FJSP. Reference [21] applied a genetic algorithm to solve FJSP with overlapping in operations. Reference [22] proposed a GA that adopted a new chromosome representation and some different strategies for crossover and mutation for FJSP. However, lack of neighborhood search procedure leads to its poor local search ability stumbling the optimization speed. TS is one of the most effective methods for solving the scheduling problem. The design of neighborhood structures determines its effectiveness. Reference [23], [24] applied a TS to solve FJSP. Reference [12] designed effective neighborhood functions of TS to solve FJSP and obtained good results. Reference [11], [13] brought up a TS method for FJSP-SDST. Reference [25] proposed a novel path-relinking algorithm based on the tabu search algorithm with back-jump tracking. However, TS merely relocates the operations in the critic path of scheduling problem, which characteristic makes it inadequacy of global search ability. Therefore, a combination of GA and TS can perform great competence in scheduling problems. Some researchers have been done to combine several algorithms to construct effective hybrid algorithms (HA) for FJSP. Reference [26] applied a hybrid genetic algorithm to deal with the sequencing problem in FJSP. Reference [27] used variable neighborhood descent to improve the search ability of GA for FJSP. Reference [28] proposed effective memetic algorithms (MAs) that combined a classic multiobjective evolutionary technique referred as NSGA-II with a novel problem-specific local search for FJSP. Reference [29] hybridized the GA and TS to solve FJSP and obtained the new best solutions for several benchmarks.

Based on the analysis of the above methods, we propose a new and effective HA-based approach named HGA-TS for FJSP-SDST-LT. We adopt a concise encoding method to represent two sub-problems: machine assignment and operation sequencing and decoding method to improve the solution quality by TS. Effective genetic operators are used to perform selection, crossover and mutation functions. In the TS, we implement new neighborhood structures and diversification functions to strengthen its local search ability. Experiments on the datasets generated from the classic benchmark instances show that HGA-TS can attain good results in short iterations.

The primary contributions are summarized as follows:

1) We formulate a mathematical model considering extra two practical characteristics upon classic FJSP.
2) We employ a hybrid algorithm HGA-TS to solve this problem effectively. It combines the exploitation ability of TS and the exploration ability of GA, which shows great performance.
3) We adopt TS utilizing the problem structure of sequence-dependent setup times and job lag times, which is efficient in performing local search.
4) We compare the global search ability of diversification operator of TS and GA in this problem, and we find diversification operator of TS is effective but GA is better in exploration.

The remaining parts of this paper are organized as follows. The description and the mathematical model of FJSP-SDST-LT are presented in Section II. In Section III, the key elements of our HGA-TS algorithm are elaborated. Computational results and analyses are summarized in Section IV. Conclusions and future work are drawn in Section V.

II. PROBLEM DESCRIPTION

For the convenience of notation, we use $[x]$ to represent set $\{1, 2, \ldots, x\}$, of which x is a positive integer. The FJSP-SDST-LT can be stated as follows. There are a set of n jobs $J = [n]$ and a set of m machines $M = [m]$. Each job i consists of a sequence of operations $O_i = \{O_{i1}, O_{i2}, \ldots, O_{id_i}\}$, where d_i denotes the number of operations that job i needs. Each operation O_{ij}, $\forall i \in [n]$, $j \in [d_i]$, has to be designated to a certain machine k from a set of compatible machines $M_{ij} \subseteq M$. For each operation O_{ij}, let p_{ijk} be the processing time on machine $k \in M_{ij}$. Stimulated by practical applications in modern manufacturing system, a setup time $s_{ii'k}$ is incurred when operations of job i and job i' are processed sequentially on machine k. Besides, $s_{ii'k}$ is only defined if the compatible machine set M_{ij} of O_{ij} and $M_{i'j'}$ of $O_{i'j'}$ have intersections. Namely, $k \in M_{ij} \cap M_{i'j'}$, if $M_{ij} \cap M_{i'j'} \neq \varnothing$. Specifically, s_{0ik} is the initial setup time for job i to be processed on machine k. In addition, $s_{ii'k}$ could be extended to $s_{iji'j'k}$ by considering the setup time incurred when operation O_{ij} and $O_{i'j'}$ are processed sequentially on machine k. The setup times satisfy the following triangular inequality:

$$s_{ii''k} \leq s_{ii'k} + s_{i'i''k}, \quad \forall\, i, i', i'' \in J,\ j \in [d_i],\ j' \in [d_{i'}],\ j'' \in [d_{i''}],\ k \in M_{ij} \cap M_{i'j'} \cap M_{i''j''} \tag{1}$$

A lag time l_{ij} is incurred in the sequential operation process order of jobs independent of machines. It does not affect the beginning processing time $st_{i'j'}$ of successive operation $O_{i'j'}$ on the same machine when $i \neq i'$. In addition, st_{ij+1} should be no less than the finishing processing time of its previous operation O_{ij} plus l_{ij}, i.e., $st_{ij+1} \geq st_{ij} + p_{ijk} + l_{ij}$.

The other assumptions considered in this paper are summarized as follows:

1) Job preemption is not permitted, i.e., each operation must be completed without interruption once started.
2) Each machine can handle only one job at a time.
3) The different operations of one job are required to be processed in a certain sequence, rather than simultaneously.
4) All jobs and machines are available at time zero.

The objective is to minimize the time required to complete all jobs, i.e., the makespan C_{max}, by assigning each job to an eligible machine and sequencing the operations on each machine.

To formulate the problem, we define two binary variables below:

$$\alpha_{ijk} = \begin{cases} 1, & \text{if } O_{ij} \text{ is assigned to machine } k \\ 0, & \text{otherwise} \end{cases}$$

$$\beta_{iji'j'} = \begin{cases} 1, & \text{if } O_{ij} \text{ is scheduled before } O_{i'j'} \\ 0, & \text{otherwise} \end{cases}$$

The mathematical model is formulated as follows:

$$\min\ C_{max} \tag{2}$$

$$\sum_{k \in M_{ij}} \alpha_{ijk} = 1,\ \forall\, i \in J,\ j \in [d_i] \tag{3}$$

$$st_{ij+1} \geq st_{ij} + \sum_{k \in M_{ij}} p_{ijk}\alpha_{ijk} + l_{ij}\ , \forall\, i \in J,\ j \in [d_i - 1] \tag{4}$$

$$st_{ij} \geq st_{i'j'} + p_{i'j'k} + s_{i'ik} - (2 - \alpha_{ijk} - \alpha_{i'j'k} + \beta_{iji'j'})H,\ \forall\,(i, i') \in J \times J,\ j \in [d_i],\ j' \in [d_{i'}],\ s.t.\ O_{ij} \neq O_{i'j'},\ k \in M_{ij} \cap M_{i'j'} \tag{5}$$

$$st_{i'j'} \geq st_{ij} + p_{ijk} + s_{ii'k} - (3 - \alpha_{ijk} - \alpha_{i'j'k} - \beta_{iji'j'})H,\ \forall\,(i, i') \in J \times J,\ j \in [d_i],\ j' \in [d_{i'}],\ s.t.\ O_{ij} \neq O_{i'j'},\ k \in M_{ij} \cap M_{i'j'} \tag{6}$$

$$C_{max} \geq st_{id_i} + \sum_{k \in M_{id_i}} p_{id_ik}\alpha_{id_ik},\ \forall\, i \in J \tag{7}$$

$$\alpha_{ijk} \in \{0, 1\},\ \forall\, i \in J,\ j \in [d_i],\ k \in M_{ij} \tag{8}$$

$$\beta iji'j' \in \{0, 1\},\ \forall\,(i, i') \in J \times J,\ j \in [d_i] \tag{9}$$

The objective (2) is to minimize the makespan C_{max}. Constraint (3) ensures that each operation is assigned to one and only one of its compatible machines. Constraint (4) ensures that operations of the same job should be processed in the given sequential order. Constraints (5) and (6) prevent the overlapping of operations on the same machine k. H stands for a large enough number. These constraints are only activated when O_{ij} and $O_{i'j'}$ are both assigned to machine k. Constraint (5) ensures that if O_{ij} is scheduled after $O_{i'j'}$, the beginning processing time of O_{ij} is later than the completion time of $O_{i'j'}$. Constraint (6) is activated when $\beta_{iji'j'} = 1$. The makespan is determined by Constraint (7). Constraint (8) and (9) ensure the decision variables are binary.

III. SOLUTION METHODOLOGY

A. WORKFLOW OF THE PROPOSED HGA-TS

In the paper, the proposed HGA-TS combines GA and TS to solve FJSP-SDT-LT. Its workflow is described in Algorithm 1:

Algorithm 1 Framework of HGA-TS

```
1: Step 1: Parameter setting for the HGA-TS;
2: Step 2: Initialization: randomly generate the initial population and set the algorithm iteration ITER: = 1;
3: Step 3: Evaluation: calculate the objective (makespan) of each individual in the population.
4: Step 4: Termination satisfaction: if the ITER reaches the max iteration or the best result has not improved for given iterations and go to Step 7; otherwise, go to Step 5;
5: Step 5: Reproduce offsprings:
6:     Step 5.1: Each time two selected parents conduct crossing over and mutation to generate two offsprings to form new population until the new population size reaches the given max size.
7:     Step 5.2: Apply TS to every offspring to improve quality.
8: Step 6: Set ITER: = ITER + 1 and go to Step 3;
9: Step 7: Output the best solution.
```

B. ENCODING AND DECODING

Chromosomes represent the solutions of the FJSP-SDST-LT. In this paper, we adopt the encoding method in [28] to encode the solution to two chromosomes which are

Procedure 1 Decoding Operator

```
Input: OS, MS
Output: Solution
1:  for $i \leftarrow 1$ to $N$ do
2:      determine the operation $O_{ij}$ in the $i$th element of OS
3:      if $j = 0$ then
4:          $as_{ij} \leftarrow s_{0ik}$
5:      else
6:          $as_{ij} \leftarrow ct_{i(j-1)} + lt_{i(j-1)}$
7:      end if
8:      determine the assigned machine $k$ of $O_{ij}$ in the $j$th position of the $i$th part in MS
9:      for $[st_k, et_k]$ in the idle period set of machine $k$ do
10:         if $\max(st_k + s_{i'ik}, as_{ij}) + p_{ijk} + s_{ii''k} \leq et_k$, $\beta i'j'ij = 1$, $\beta iji''j'' = 1$ then
11:             $st_{ij} \leftarrow \max(st_k + s_{i'ik}, as_{ij})$
12:             insert $O_{ij}$ into machine $k$
13:             the idle period of machine $k$
14:             break
15:         end if
16:     end for
17:     $ct_{ij} \leftarrow st_{ij} + p_{ijk}$
18: end for
```

employed to correspond to the two subproblems. The first is for operation sequence (OS) and the second is for machine assignment (MS).

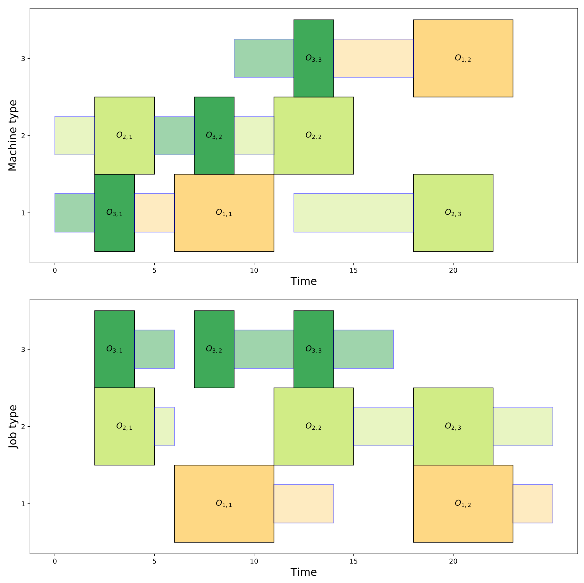
Every number in OS represents a job type. Each job type i occurs d_i times. The corresponding operation type j refers to the jth appearance of job i by scanning the OS from left to right. Thus, the length of OS chromosome is equal to the total number of operations, i.e., $\sum_{i=1}^{n} d_i$. The significant advantage of OS is that any permutation can be decoded to a feasible solution.

The MS represents the selected machines of each operation. It is separated into n parts. The ith part contains d_i operations of job i. So every position in MS denotes the selected machine for operation O_{ij}, according to its located part and the jth position in it. The length of MS corresponds to the length of OS.

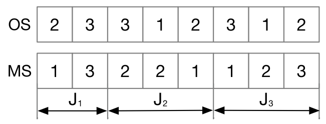
The decoding method aims to decode chromosomes into a solution, which is conducive to apply TS to improve its quality. A solution contains the scheduling information of each operation O_{ij} which includes its assigned machine k, st_{ij}, and p_{ijk}. We define four additional variables to explain the decoding method:

as_{ij}	the arriving time of O_{ij}
ct_{ij}	the completion time of O_{ij}
st_k	the starting time of idle period of machine k
et_k	the ending time of idle period of machine k

Based on the decoding method, the information of each operation O_{ij} including st_{ij}, ct_{ij} and its assigned machine k are determined. Thus, we get an intact schedule for the work shop. We exemplify the relation between a solution and its chromosomes by Fig. 1. In the top subfigure of Fig. 1(a), the length of semi-transparent bar is equal to the sequence-dependence setup times. In the bottom subfigure of Fig. 1(a), the length of semi-transparent bar is equal to the job lag times. Fig. 1(b) represents the encoded OS and MS chromosomes corresponding to Fig. 1(a). The permutation of OS is equal to the permutation of $O_{21}, O_{31}, O_{32}, O_{11}, O_{22}, O_{33}, O_{12}, O_{23}$. The permutation of MS represents the machine assignment information which is (O_{11}, M_1), (O_{12}, M_3), (O_{21}, M_2), (O_{22}, M_2), (O_{23}, M_1), (O_{31}, M_1), (O_{32}, M_2), (O_{33}, M_3) and they can be decoded into the solution represented in Fig. 1(a).



(a) Gantt chart solution of example instance



(b) Chromosomes of example instance

FIGURE 1. **Example of encoding and decoding.**

C. GENETIC OPERATORS

Proper genetic operators contribute to generating excellent offspring. We adapt three operators from [29] which are selection, crossover, and mutation.

1) SELECTION

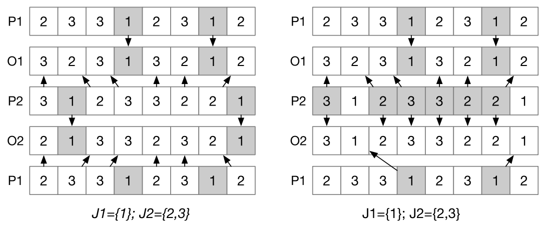
The selection operator is used to select individuals according to the fitness. We employ the elitist selection operator and the tournament operator. In this paper, the elitist operator selects N individuals with the shortest makespan as parents. The tournament operator chooses b (in this paper $b = 2$) individuals from the population each time and selects the individual with the shortest makespan as a

Procedure 2 POX Operator

```
1: Step1: Divide job set $J$ into two groups $J1$ and $J2$ randomly;
2: Step2: Any element of OS in $P1$ which belongs to $J1$ are relocated to the same position of OS in $O1$ and deleted from $P1$. Any element of OS in $P2$ which belongs to $J2$ are relocated to the same position of OS in $O2$ and deleted from $P2$;
3: Step3: The remaining elements of OS in $P2$ are relocated to the remaining empty positions of OS in $O1$ sequentially. The remaining elements of OS in $P1$ are relocated to the remaining empty positions of OS in $O2$ sequentially.
```

Procedure 3 JBX Operator

```
1: Step1: Divide the job set $J$into two groups $J1$ and $J2$ randomly;
2: Step2: Any element of OS in $P1$ which belongs to $J1$ are allocated to the same position of OS in $O1$ and deleted from $P1$. Any element of OS in $P2$ which belongs to $J2$ are allocated to the same position of OS in $O2$ and deleted from $P2$;
3: Step3: The remaining elements of OS in $P2$ are allocated to the remaining empty positions of OS in $O1$ sequentially. The remaining elements of OS in $P1$ are allocated to the remaining empty positions of OS in $O2$ sequentially.
```



(a) POX Crossover (b) JBX Crossover

FIGURE 2. Crossover operators for OS.

parent until the parent set reaches the maximum of population size.

2) CROSSOVER

We adopt the precedence operation crossover (POX) and the job-based crossover (JBX) for the OS. Two operators are employed at 50%. We denote two parents as $P1$ and $P2$ and two offspring as $O1$ and $O2$. The procedure of POX is described in Procedure 2 and is illustrated in Fig. 2(a). The procedure of JBX is decribed in Procedure 3 and is illustrated in Fig. 2(b). In addition, we adopt a two-point crossover for the MS. The procedure is described in Procedure 4 and is illustrated in Fig. 3.

Procedure 4 Two-Point Crossover Operator

```
1: Step1: Randomly choose two non-repetitive points $p1$ and $p2$ from 1 to $N$ ($p1 < p2$);
2: Step2: Elements of MS which positions are in$[p1, p2]$ in $P1$ are allocated to the same position of MS in $O2$. Remaining elements are allocated to the same position of MS in $O1$. Elements of MS which positions are in$[p1, p2]$ in $P2$ are allocated to the same position of MS in $O1$. Remaining elements are allocated to the same position of MS in $O2$.
```

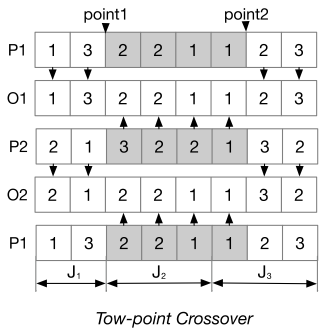


FIGURE 3. Two-point crossover for MS.

Procedure 5 Swapping Mutation Operator

```
1: Step1: Select two non-repetitive positions $p1$ and $p2$ randomly in the OS of $P1$;
2: Step2: Swap the element on $p1$ and $p2$ and generate $O1$.
```

Procedure 6 Neighborhood Mutation Operator

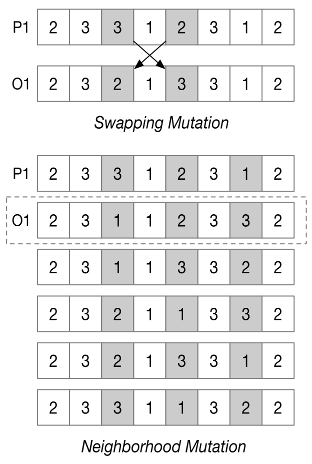
```
1: Step1: Select three non-repetitive elements randomly in the OS of $P1$;
2: Step2: Generate all the permutations of these three elements as neighborhood OS;
3: Step3: Choose one OS from neighborhood randomly, and generate $O1$ by setting it as the current OS.
```

3) MUTATION

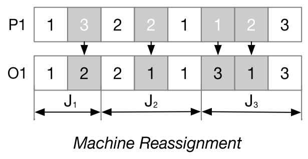
We adopt the swapping mutation operator and the neighborhood operator for OS and each operator is selected at 50%. We denote the individual before mutation as $P1$ and the individual after mutation as $O1$. The swapping operator is described in Procedure 5. The neighborhood mutation operator is described in Procedure 6. Both of them are illustrated in Fig. 4(a). For the MS, a machine reassignment operator is described in Procedure 7 and is illustrated in Fig. 4(b).

D. PROBLEM STRUCTURE

Before introducing the TS, we first describe our problem structure to construct neighborhood structure efficiently.



(a) Mutation operators for OS



(b) Mutation operator for MS

FIGURE 4. Mutation operators.

Procedure 7 Machine Reassignment Operator

```
1: Step1: Select $h$ positions in the MS of $P$ (in this paper, $h = N/2$);
2: Step2: Get the corresponding operation of each position. Choose the other machine randomly from its eligible machine set as the current machine assignment.
```

In this paper, we utilize the concept of disjunctive graph $G = (O, A, E)$. Set O contains two fictitious nodes 0 and $*$, which represent the origin and the destination of the schedule. Thus, we have $O = \{0, 1, \ldots, N, *\}$. The relation between a node u and operation O_{ij} is that $u = \sum_{h=1}^{i-1} d_h + j$. The weight of node is equal to its processing time. Operation sequence relations within the same job are modeled by conjunctive arcs in set A as follows:

$$A = \bigcup_{i=1}^{n} \{(O_{ij}, O_{i(j+1)}) : j \in [d_i - 1]\} \\ \cup \{(O_{id_i}, *)\} \cup \{(0, O_{i0})\} \tag{10}$$

The length of arc in A between O_{ij} and $O_{i(j+1)}$ is equal to the lag time lt_{ij} of O_{ij}. Sequence relations within the same machine are modeled by disjunctive arcs in set E as follows:

$$E = \bigcup_{k=1}^{m} \{[O_{ij}, O_{i'j'}] : m_{ij} = m_{i'j'} = k\} \\ \cup \{[0, O_{ij}]\} \cup \{[O_{ij}, *]\} \tag{11}$$

The length of arc in E between O_{ij} and $O_{i'j'}$ on machine k is equal to the setup time $s_{ii'k}$. Transforming E into conjunctive graph will generate permutation Ω in the following:

$$\Omega = \bigcup_{k=1}^{m} \{(O_{ij}, O_{i'j'}) : m_{ij} = m_{i'j'} = k\} \\ \cup \{(0, O_{ij})\} \cup \{(O_{ij}, *)\} \tag{12}$$

For the sake of constructing neighborhood structure efficiently, we introduce some definitions. Specifically, a path from nodes u to v contains a finite set of sequential nodes and conjunctive arcs between u and v. The length of a path $L(u, v)$ is equal to the sum of the weights of nodes and the length of the arcs on the path [24]. A critical path refers to a longest path between node 0 and node $*$ and its length $L(0, *)$ is the makespan. Next, we introduce resource extending equation for each operation. We denote node u's head as r_u which can be viewed as a longest path from 0 to u without u's node weight and its tail as q_u which can be viewed as a longest path from u to $*$. Practically, r_u can be seen as the earliest beginning time of u and $[L(0, *) - q_u]$ can be seen as the latest completion time of u. We define r_u^J as the path generated from the previous operation $[u - 1]$ on its job routing and r_u^M as the path generated from the previous operation w on its machine routing. We define q_u^J as the path generated from the successor node $[u + 1]$ on its job routing and q_u^M as the path from the successor node v on its machine routing. Specifically, the calculation is demonstrated as follows:

$$r_u = \max\{r_u^J, r_u^M\}, \tag{13}$$
$$r_u^J = r_{u-1} + p_{u-1} + l_{u-1}\ (u-1, u) \in A, \tag{14}$$
$$r_u^M = r_w + p_w + s_{wu}\ (w, u) \in \Omega, \tag{15}$$
$$q_u = \max\{q_u^J, q_u^M\}, \tag{16}$$
$$q_u^J = q_{u+1} + p_{u+1} + l_u\ (u, u+1) \in A, \tag{17}$$
$$q_u^M = q_v + p_v + s_{uv}\ (u, v) \in \Omega, \tag{18}$$

The makespan is defined by

$$C_{max} = \max_{u \in O}\{r_u + q_u + p_u\} \tag{19}$$

We denote operation u on the critical path as critical operation, which satisfies $r_u + q_u + p_u = C_{max}$.

E. TABU SEARCH ALGORITHM

In order to improve the quality of the offspring reproduced by genetic operators, we adapt the TS from [13] for FJSP-SDT-LT. TS contains four parts: neighborhood structure with feasibility checks, preliminary neighbor evaluations, tabu list, and diversification.

1) FEASIBILITY CHECK

Before constructing relocating neighborhood structure, we first guarantee that the directed graph does not contain cycles to acquire feasible solutions. In this paper, we develop a proposition for FJSP-SDST-LT based on [24] as follows:

Theorem 1: For FJSP-SDST-LT, sequencing operation u between v and w does not lead to infeasible solutions if:

1) $r_v < r_{u+1} + p_{u+1} + \min_{k \in M_{u+1} \cap M_v} s_{(u+1)vk}$ with $(u, u+1) \in A$, $[u, v] \notin A$.

2) $r_w + p_w + \min_{k \in M_w \cap M_{u-1}} s_{w(u-1)k} > r_{u-1}$ with $(u-1, u) \in A$, $[w, u] \notin A$.

Proof: Removing operation u from the graph means breaking arcs $(a, u), (u, b) \in \Omega$, generating graph G^- which does not contain cycles. Inserting u between v and w means adding arcs (v, u) and (u, w) which only generates cycles if path $(v, u, u+1, \ldots, v)$ or $(u-1, u, w, \ldots, u-1)$ exist. In this paper, we consider the minimum setup time between operations in the path. The first condition guarantees that $u+1$ does not appear before v and thus no cycle $(v, u, u+1, \ldots, v)$ will be generated, while the second condition guarantees that no cycle $(u-1, u, w, \ldots, u-1)$ will be generated as well. Therefore the proposition prevents infeasible operation move and ensures feasibility.

2) NEIGHBORHOOD STRUCTURE

We assume that arcs $(a, u), (u, b), (v, w) \in \Omega$ are in the current graph and u is on machine m_u. Neighborhood is constructed by relocating u from its current position to another position based on the feasibility propositions. According to [12], only operations on the critical path CP determine the makespan directly, i.e., $C_{max} = r_u + p_u + q_u, u \in CP$. Thus, we relocate operations on one critical path out of efficiency. The relocating operator is adapted from [13] and described in Procedure 8.

Procedure 8 Relocating Operator

```
1: Find a critical path $CP$
2: for operation $u$ in $P$ do
3:     if $u$ satisfies $C_{max} = r_u^J + p_u + q_u^J$ then
4:         continue
5:     end if
6:     for machine $k$ in $M_u$ do
7:         for position $i$ in machine $k$ do
8:             if $u$ satisfies $C_{max} = r_u^J + p_u + q_u^M$ then
9:                 neglect positions before $u$ on machine $m_u$
10:            end if
11:            if $u$ satisfies $C_{max} = r_u^M + p_u + q_u^J$ then
12:                neglect positions after $u$ on machine $m_u$
13:            end if
14:            if $u$ satisfies feasibility conditions by inserting on
               position $i$ then
15:                generate a new neighbor and add into neigh-
                   borhood
16:            end if
17:        end for
18:    end for
19: end for
20: Output neighborhood
```

3) NEIGHBOR EVALUATION

In order to evaluate neighborhood efficiently instead of calculating the makespan from scratch, we employ the method in [13] to calculate the lower bounds on the new makespan. We denote P_v and S_w as sets containing all predecessors of v and successors of w. If u is inserted between operation v and w and $r_v > r_u$, the lower bound of the new makespan is:

$$LB_1 = \max\{r_u^J; \hat{r_v} + p_v + s_{vu}\} + p_u + \max\{q_u^J; s_{uw} + q_w + p_w\} \quad (20)$$

where

$$\hat{r_v} = \begin{cases} r_v, & u \notin P_v \\ \max\{r_b^J; r_u^M - s_{au} + s_{ab}\} + r_v - r_b, & u \in P_v \end{cases} \quad (21)$$

If u is inserted before w and $r_v > r_w$, the lower bound of the new makespan is:

$$LB_2 = \max\{r_u^J; r_v + p_v + s_{vu}\} + p_u + \max\{q_u^J; s_{uw} + \hat{q_w} + p_w\} \quad (22)$$

where

$$\hat{q_w} = \begin{cases} q_w, & u \notin S_w \\ \max\{q_a^J; q_u^M - s_{ub} + s_{ab}\} + q_w - q_a, & u \in S_w \end{cases} \quad (23)$$

4) TABU LIST

When relocating operation u from machine k to machine k', we store (u, k) and (u, k') in tabu list.

5) DIVERSIFICATION

The neighborhood structure focusing on critical operations neglects optimizing the whole scheduling structure. For this case, we employ a diversification structure to help jump out of local optimum. When no improvement of the makespan reaches a certain number of iterations, diversification is activated. Operation u which satisfies $C_{max} = r_u^J + p_u + q_u^J$ and non-critical operations are relocated to another position randomly. The diversification is repeated for prescribed iterations. When the number of diversification reaches a certain number, we generate the initial solution randomly from scratch as the current solution.

IV. NUMERICAL RESULTS AND DISCUSSION

A. EXPERIMENTAL DESIGN

To evaluate the performance of the proposed HGA-TS, we conduct experiments on two classes of FJSP-SDST-LT instances derived from benchmarks in [24], [30] respectively. We denote them as Set 1 and Set 2. The algorithms are coded in Java and implemented on a computer with 2.2 GHz Intel Core i7 and 16 GB of RAM memory. The MILP is solved by IBM ILOG CPLEX 12.8.0. The parameters including HGA-TS, pure GA, and TS for these instances are shown in Tables 1 and 2.

Set 1 consists of 10 problem instances where the number of jobs ranges from 10 to 20, the number of machines ranges from 6 to 15, the number of operations for each job ranges from 5 to 15, and the maximum number of equivalent machines per operation ranges from 3 to 6. Set 2 consists of 18 test problems where the number of jobs ranges from 10 to 20, the number of machines ranges from 5 to 10, and the

TABLE 1. The HGA-TS parameters.

Parameters	Descriptions	
$Popsize$	The size of the population	200
$maxIter$	The max iteration	100
$maxTN$	The termination iteration without improvement	20
p_e	The elite selection proportion	0.01
p_c	Crossover probability	0.8
p_m	Mutation probability	0.1
ρ	The maximum iteration factor of TS	800

TABLE 2. The GA and TS parameters.

Parameters	Descriptions	
$Popsize$	The size of the population	200
$maxIter$	The max iteration	10000
$maxTN$	The termination iteration without improvement	500
p_e	The elite selection proportion	0.01
p_c	Crossover probability	0.8
p_m	Mutation probability	0.1
$maxIterTs$	The maximum iteration of TS	10000
dt	Diversification tenure	3
dat	Diversification activation tenure	500
rs	Restart search	2000

number of operations for each job ranges from 5 to 25. The set of machines capable of processing an operation is constructed by letting a machine be in that set with a probability ranging from 0.1 to 0.5.

B. RESULTS AND DISCUSSIONS

Tables 3 and 4 show the experiment results (C_{max}) of HGA-TS and MILP (time limit is 4 hours) on Set 1 and Set 2. The results marked by "$*$" are the best results. The results marked by "$-$" indicate that MILP is unable to find a feasible solution in the limited time. MRE (%) column shows a comparison between MILP and our HGA-TS, which means the percentage improvement calculated as:

$$MRE = \frac{C_{max}^{MILP} - C_{max}^{HGA-TS}}{C_{max}^{MILP}} \tag{24}$$

The result shows that MILP is not qualified in solving large-scale instances and HGA-TS can find feasible solutions better than MILP on two classes of data set, improving the upper bound by 9.7% and 15.8% respectively. It shows the great effectiveness of HGA-TS in solving FJSP-SDST-LT.

Percentage improvement is illustrated in Fig. 5 and Fig. 6 which shows improvement among heuristic algorithms compared with HGA-TS on Set 1 and Set 2. The heuristic algorithms are obtained by algorithms proposed in Section III that includes GA (pure genetic algorithm without tabu search algorithm), TS1 (tabu search algorithm with diversification operator), TS2 (tabu search algorithm without diversification operator). The results show that our HGA-TS outperforms

TABLE 3. Computational results on set 1.

Instance	n×m	MILP	HGA-TS	MRE (%)
sl01	10×6	83	81*	2.41%
sl02	10×6	70	65*	7.14%
sl03	15×8	261	252*	3.45%
sl04	15×8	113	109*	3.54%
sl05	15×4	293	269*	8.19%
sl06	10×15	159	141*	11.32%
sl07	20×5	257	210*	18.29%
sl08	20×10	673	639*	5.05%
sl09	20×10	492	388*	21.14%
sl10	20×15	359	300*	16.43%

TABLE 4. Computational results on set 2.

Instance	n×m	MILP	HGA-TS	MRE
sl01a	10×5	3068	2689*	12.35%
sl02a	10×5	2645	2418*	8.58%
sl03a	10×5	2718	2416*	11.11%
sl04a	10×5	2866	2667*	6.94%
sl05a	10×5	2614	2400*	8.19%
sl06a	10×5	2820	2395*	15.07%
sl07a	15×8	2820	2479*	12.09%
sl08a	15×8	2953	2253*	23.70%
sl09a	15×8	3378	2245*	33.54%
sl10a	15×8	3039	2493*	17.97%
sl11a	15×8	-	2255*	-
sl12a	15×8	-	2222*	-
sl13a	20×10	2895	2468*	14.75%
sl14a	20×10	2989	2351*	21.34%
sl15a	20×10	-	2352*	-
sl16a	20×10	3082	2464*	20.05%
sl17a	20×10	-	2337*	-
sl18a	20×10	-	2325*	-

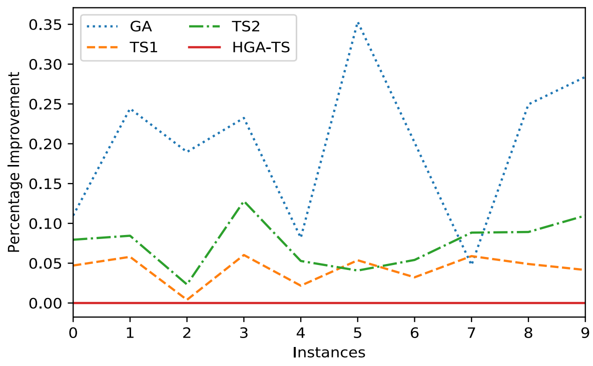


FIGURE 5. Comparison among heuristic algorithms on set 1.

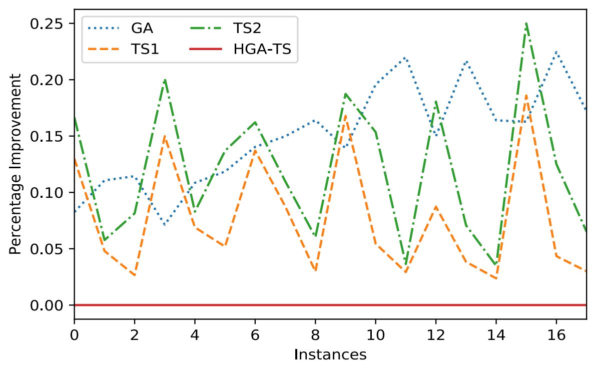


FIGURE 6. Comparison among heuristic algorithms on set 2.

TABLE 5. The levels of GA parameters.

Parameters	Level 1	Level 2	Level 3	Level 4
$Popsize$	40	60	80	100
$maxIter$	50	100	150	200

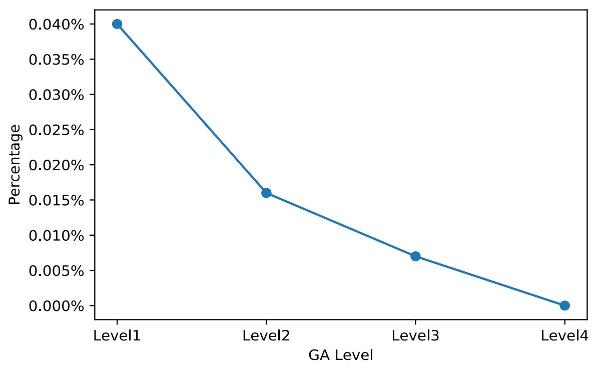


FIGURE 7. Impact of GA parameter levels on HGA-TS.

other heuristic algorithms on data set of short processing time and long processing time at the same time. Meanwhile, it is notable that TS1 outperforms TS2 and GA in almost every instance but lags behind HGA-TS, which shows the diversification operator can help the solution escape local optimality to some extent, but its global search ability is weaker than GA.

We also show the impact of GA parameter levels on HGA-TS in Fig. 7. GA parameter levels are showed in Table 5, which include four levels ranging from low population size and iteration to high population size and iteration. Fig. 7 shows the average percentage improvement of Level 4 compared with other levels. It illustrates that with the level increasing, the GA impact on optimization decreases, and a low level of GA parameters is enough to find a good solution. In consequence, the computational results demonstrate that our HGA-TS dominates its components and MILP, showing great ability in solving FJSP-SDST-LT even with low population size and iteration. The reasons are demonstrated as follows: First and foremost, we use two selection operators, the elite selection focusing on the best individuals of the population, and tournament selection balancing the quality and diversity of the selected individuals. The crossover and mutation operators designed for OS and MS respectively ensure its effective exploration. With the local improvement of TS, the quality of individuals can get improved further. Therefore, our proposed algorithm has distinctive advantages over other algorithms.

V. CONCLUSION

In this paper, by employing the global diversification ability of GA and the local improvement ability of TS, we propose a hybrid genetic algorithm HGA-TS for FJSP-SDST-LT. Experiments are conducted on two classes of data set generated from two famous benchmark instances. The computational results show HGA-TS outperforms other heuristic algorithms and can find better upper bounds than MILP, proving its effectiveness in solving FJSP-SDST-LT. For future studies, multi-objective optimization is required for a practical scheduling environment in which the objective not only includes makespan but tardiness and machine load balance. Besides, various practical constraints such as work schedule and resource limitation could be of interest.

REFERENCES

[1] M. Pinedo and K. Hadavi, *Scheduling: Theory, Algorithms and Systems Development*. Berlin, Germany, 1992, pp. 35–42.

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

[3] A. Allahverdi, C. T. Ng, T. C. E. Cheng, and M. Y. Kovalyov, "A survey of scheduling problems with setup times or costs," *Eur. J. Oper. Res.*, vol. 187, no. 3, pp. 985–1032, 2008.

[4] I. A. Chaudhry and A. A. Khan, "A research survey: Review of flexible job shop scheduling techniques," *Int. Trans. Oper. Res.*, vol. 23, no. 3, pp. 551–591, May 2016.

[5] C. A. Méndez, J. Cerdá, I. E. Grossmann, I. Harjunkoski, and M. Fahl, "State-of-the-art review of optimization methods for short-term scheduling of batch processes," *Comput. Chem. Eng.*, vol. 30, nos. 6–7, pp. 913–946, May 2006.

[6] M. Saidi-Mehrabad and P. Fattahi, "Flexible job shop scheduling with tabu search algorithms," *Int. J. Adv. Manuf. Technol.*, vol. 32, nos. 5–6, pp. 563–570, Mar. 2007.

[7] S. Nourali and N. Imanipour, "A particle swarm optimization-based algorithm for flexible assembly job shop scheduling problem with sequence dependent setup times," *Scientia Iranica*, to be published.

[8] F. M. Defersha, "An efficient two-stage genetic algorithm for a flexible job-shop scheduling problem with sequence dependent attached/detached setup, machine release date and lag-time," *Ind. Eng.*, vol. 147, Sep. 2020, Art. no. 106605.

[9] A. Rossi and G. Dini, "Flexible job-shop scheduling with routing flexibility and separable setup times using ant colony optimisation method," *Robot. Comput.-Integr. Manuf.*, vol. 23, no. 5, pp. 503–516, Oct. 2007.

[10] A. Rossi, "Flexible job shop scheduling with sequence-dependent setup and transportation times by ant colony with reinforced pheromone relationships," *Int. J. Prod. Econ.*, vol. 153, pp. 253–267, Jul. 2014.

[11] T. F. Abdelmaguid, "A neighborhood search function for flexible job shop scheduling with separable sequence-dependent setup times," *Appl. Math. Comput.*, vol. 260, pp. 188–203, Jun. 2015.

[12] M. Mastrolilli and L. M. Gambardella, "Effective neighbourhood functions for the flexible job shop problem," *J. Scheduling Banner*, vol. 3, no. 1, pp. 3–20, 2000.

[13] L. Shen, "Solving the flexible job shop scheduling problem with sequence-dependent setup times," *Eur. J. Oper. Res.*, vol. 265, no. 2, pp. 503–516, 2018.
[14] D. Kress, D. Müller, and J. Nossack, "A worker constrained flexible job shop scheduling problem with sequence-dependent setup times," *OR Spectr.*, vol. 41, no. 1, pp. 179–217, Mar. 2019.
[15] S. Nourali, N. Imanipour, and M. R. Shahriari, "A mathematical model for integrated process planning and scheduling in flexible assembly job shop environment with sequence dependent setup times," *Int. J. Math. Anal.*, vol. 6, nos. 41–44, pp. 2117–2132, 2012.
[16] F. M. Defersha and M. Chen, "Jobshop lot streaming with routing flexibility, sequence-dependent setups, machine release dates and lag time," *Int. J. Prod. Res.*, vol. 50, no. 8, pp. 2331–2352, Apr. 2012.
[17] J. M. Novas, "Production scheduling and lot streaming at flexible job-shops environments using constraint programming," *Comput. Ind. Eng.*, vol. 136, pp. 252–264, Oct. 2019.
[18] F. Pezzella, G. Morganti, and G. Ciaschetti, "A genetic algorithm for the flexible job-shop scheduling problem," *Comput. Oper. Res.*, vol. 35, no. 10, pp. 3202–3212, 2008.
[19] L. De Giovanni and F. Pezzella, "An improved genetic algorithm for the distributed and flexible job-shop scheduling problem," *Eur. J. Oper. Res.*, vol. 200, no. 2, pp. 395–408, Jan. 2010.
[20] J. C. Chen, C.-C. Wu, C.-W. Chen, and K.-H. Chen, "Flexible job shop scheduling with parallel machines using genetic algorithm and grouping genetic algorithm," *Expert Syst. Appl.*, vol. 39, no. 11, pp. 10016–10021, Sep. 2012.
[21] Y. Demir and S. K. Işleyen, "An effective genetic algorithm for flexible job-shop scheduling with overlapping in operations," *Int. J. Prod. Res.*, vol. 52, no. 13, pp. 3905–3921, Jul. 2014.
[22] I. Driss, K. N. Mouss, and A. Laggoun, "A new genetic algorithm for flexible job-shop scheduling problems," *J. Mech. Sci. Technol.*, vol. 29, no. 3, pp. 1273–1281, Mar. 2015.
[23] J. Hurink, B. Jurisch, and M. Thole, "Tabu search for the job-shop scheduling problem with multi-purpose machines," *OR Spektrum*, vol. 15, no. 4, pp. 205–215, Dec. 1994.
[24] S. Dauzère-Pérès and J. Paulli, "An integrated approach for modeling and solving the general multiprocessor job-shop scheduling problem using tabu search," Tech. Rep., 1997.
[25] S. Jia and Z. H. Hu, "Path-relinking tabu search for the multi-objective flexible job shop scheduling problem," *Comput. Oper. Res.*, vol. 47, no. 1, pp. 11–26, Jul. 2014.
[26] N. Zribi, I. Kacem, A. E. Kamel, and P. Borne, "Assignment and scheduling in flexible job-shops by hierarchical optimization," *IEEE Trans. Syst., Man, Cybern. C, Appl. Rev.*, vol. 37, no. 4, pp. 652–661, Jul. 2007.
[27] J. Gao, L. Sun, and M. Gen, "A hybrid genetic and variable neighborhood descent algorithm for flexible job shop scheduling problems," *Comput. Oper. Res.*, vol. 35, no. 9, pp. 2892–2907, 2008.
[28] L. Gao, C. Peng, C. Zhou, and P. Li, *Solving Flexible Job Shop Scheduling Problem Using General Particle Swarm Optimization*, 2006.
[29] X. Li and L. Gao, "An effective hybrid genetic algorithm and tabu search for flexible job shop scheduling problem," *Int. J. Prod. Econ.*, vol. 174, pp. 93–110, Apr. 2016.
[30] P. Brandimarte, "Routing and scheduling in a flexible job shop by tabu search," *Ann. Oper. Res.*, vol. 41, no. 1, pp. 157–183, Sep. 1993.

YILUN WANG was born in Yantai, Shandong, China, in 1997. He received the B.S. degree in logistics engineering from Central South University, Changsha, China, in 2019. He is currently pursuing the M.S. degree with Nanjing University, Nanjing, China.

His current research interests include optimization in job scheduling problem and vehicle routing problem.

QIANWEN ZHU was born in Chizhou, Anhui, China, in 1997. She received the B.S. degree in computer science from Chongqing University, Chongqing, China, in 2019. She is currently pursuing the M.S. degree with Nanjing University, Nanjing, China.

Her current research interests include combinatorial optimization algorithms in job scheduling problem, container relocation problem, and vehicle routing problem.