A Two-Individual Based Evolutionary Algorithm for the Flexible Job Shop Scheduling Problem

Abstract

Population-based evolutionary algorithms usually manage a large number of individuals to maintain the diversity of the search, which is complex and time-consuming. In this paper, we propose an evolutionary algorithm using only two individuals, called master-apprentice evolutionary algorithm (MAE), for solving the flexible job shop scheduling problem (FJSP). In order to ensure the diversity and the quality of the evolution, MAE integrates a tabu search procedure, a recombination operator based on path relinking using a novel distance definition, and an effective individual updating strategy, taking into account the multiple complex constraints of FJSP. Experiments on 178 public instances show that MAE achieves highly competitive results in terms of both solution quality and computational efficiency compared with the state-of-the-art algorithms. Specifically, it improves the previous best known results for 47 instances while matching the best known results on all except 5 of the remaining instances with reasonable computational time.

1 Introduction

The job shop scheduling problem (JSP) is a strongly NP-hard problem [Garey *et al.*, 1976]. In this problem, there are a set of jobs $J = \{J_1, \ldots, J_n\}$ that must be processed on a set $M = \{M_1, \ldots, M_m\}$ of machines. Each job $J_i, i = 1, \ldots, n$, consists of n_i operations $O_i = \{o_{i1}, \ldots, o_{in_i}\}$ that should be sequentially processed. Besides, each operation o_{ij} requires uninterrupted and exclusive use of its assigned machine for its whole processing time. The flexible job shop scheduling problem (FJSP) is an extension of JSP by allowing an operation o_{ij} to be processed on one of a set of candidate machines $M(o_{ij}) \subseteq M$. The processing time of operation o_{ij} on machine $M_k \in M(o_{ij})$ is denoted by $t_{o_{ij}k}$. The problem is to assign each operation to a machine and to order the operations on the machines, such that the maximal completion time of all jobs (i.e., makespan) is minimized.

Since FJSP was introduced by [Brucker and Schlie, 1991], a large number of methods for solving it have been proposed in the literature. Among them we cite several exact approaches: A discrete-time integer programming based on Lagrangian relaxation method proposed by [Thomalla, 2005], a mixed-integer linear programming model proposed by [Özgüven *et al.*, 2010] with routing and process plan flexibility, and a mixed-integer linear optimization model combined with a branch and bound algorithm proposed by [Hansmann *et al.*, 2014]. Other exact methods based on mixed-integer linear programming can be found in [Gomes *et al.*, 2013; Roshanaei *et al.*, 2013].

For large FJSP instances, exact methods are ineffective. Therefore, various metaheuristic algorithms have been employed. [Brandimarte, 1993] proposed a hybrid tabu search algorithm with several dispatching rules. [Mastrolilli and Gambardella, 2000] proposed local search techniques and developed two neighborhood functions to improve the tabu search algorithm of [Dauzère-Pérès and Paulli, 1997]. [Pezzella *et al.*, 2008] proposed operation minimum processing time heuristic and global minimum processing time heuristic. [Gao *et al.*, 2008] proposed a hybrid genetic algorithm (HGA) with a variable neighborhood descent (VND) algorithm. Recent approaches include the climbing depth-bounded discrepancy search (CDDS) algorithm [Hmida *et al.*, 2010], iterative flattening search algorithm [Oddi *et al.*, 2011], hybrid harmony search and large neighborhood search (HHS/LNS), hybrid differential evolution algorithm (HDE-N2) [Yuan and Xu, 2013a; 2013b], hybrid genetic algorithm combined with tabu search (GA+TS) [Ángel González *et al.*, 2013], parallel approach with double-level metaheuristic algorithm (TSBM^{2}h) [Bożejko *et al.*, 2010], modular genetic algorithm with repairing heuristics (MGARH) [Gutiérrez and García-Magariño, 2011], scatter search with path relinking (SSPR) [González *et al.*, 2015], partial swarm optimization (PSO) [Li *et al.*, 2010], artificial bee colony (ABC) [Wang *et al.*, 2012], estimation of distribution algorithm (EDA) [Wang *et al.*, 2013], harmony search algorithm (HS) [Gao *et al.*, 2016], hybrid genetic algorithm with tabu search (HA) [Li and Gao, 2016], genetic tabu search (HGTS) [Palacios *et al.*, 2015], and multi-start multi-level evolutionary local search (GRASP-mELS) [Kemmoé-Tchomté *et al.*, 2017]. Although none of these approaches dominates the others in terms of the solution quality and computational efficiency for all the benchmarks, CDDS, HDE-N2, SSPR, HGTS, HA, and GRASP-mELS show the best performance among them.

Population-based evolutionary algorithms have good performance for tackling FJSP. However, they suffer from the drawback of managing a large population to maintain the diversity of the search [Ansotegui *et al.*, 2015]. In this paper, we propose an evolutionary algorithm using only two individuals, called master-apprentice evolutionary (MAE) algorithm, for solving FJSP, inspired from *HEAD* [Moalic and Gondran, 2017], the only previous evolutionary algorithm based on two individuals to the best of our knowledge. *HEAD* is for solving the k-coloring problem. The particularity of the k-coloring problem is that its constraints are very simple, whereas FJSP has multiple complex constraints. Consequently, *HEAD* and MAE have to be very different: *HEAD* uses the greedy partition crossover to generate the child solutions, because it works in the space of infeasible solutions to search for a feasible k-coloring, whereas MAE uses a recombination operator based on path relinking with a novel distance definition to generate child solutions, because it works in the space of feasible solutions to search for an optimal feasible solution. In fact, a crossover operator often generates infeasible child solutions for FJSP, and repairing these solutions then results in poor solutions, whereas a recombination operator based on path relinking can be more easily controlled to generate feasible child solutions. Besides, the diversity in the search of *HEAD* is also maintained by the crossover operator, whereas MAE maintains the diversity by directly replacing one individual with a random feasible solution as soon as the two individuals become close to each other.

The remaining part of this paper is organized as follows: Section 2 presents the proposed MAE algorithm. Section 3 compares MAE with the state-of-the-art algorithms and analyzes the key features of MAE to identify its success factors. Section 4 concludes the paper.

2 Master-Apprentice Evolutionary Algorithm

The idea of the master-apprentice evolutionary algorithm originates from social activities where apprentices gain knowledge from their masters. When two apprentices (individuals) evolve for a given number of generations (a cycle), they become masters and share much similarity. Therefore, when a cycle ends, one apprentice is replaced with the master in the previous cycle to continue the evolution, so as to absorb the essence of the history (the previous cycle). That is why we call this algorithm master-apprentice evolutionary algorithm.

Besides, two-individual based evolution mechanism is a unique feature of MAE. In many cases of human activities, two persons may be easy to achieve a goal since it has collaboration and is easy to manage, which is different from a single person or an entire population. By managing two individuals using an effective individual updating strategy, MAE can achieve a better trade-off between diversification and search efficiency. In this section, we first present the general architecture of MAE and then present its different components.

2.1 Main scheme of MAE

MAE follows the basic framework of the evolutionary algorithms [Lü *et al.*, 2010; Yu *et al.*, 2013; Lau *et al.*, 2013]. Its diagram is depicted in Fig. 1 and its general architecture is described in Algorithm 1. The algorithm has three main components: The Init() function to generate a random solution, the tabu search procedure TS(S) to improve the solution S, and the path relinking based recombination operator to generate two child solutions. The generations are divided into cycles of length p, where p is an integer parameter. The best solution in the current (previous) cycle is stored in S_c^* (S_p^*). At the beginning, MAE generates two random solutions S_1 and S_2. Then, at each generation, it applies the path relinking based recombination operator on S_1 and S_2 to generate two child solutions $S_1^{'}$ and $S_2^{'}$, which are then optimized by the tabu search procedure to become new S_1 and S_2. If the new S_1 or S_2 is better than S_c^*, then S_c^* is updated. At the end of each cycle, S_1 is replaced by the best solution S_p^* found in the previous cycle, S_p^* is replaced by S_c^*, and S_c^* is set to be a random solution, before starting the next cycle. As soon as S_1 is close to S_2, S_2 is replaced with a random solution to ensure the diversity of the search. Finally, the best solution S^* found during the search is returned.

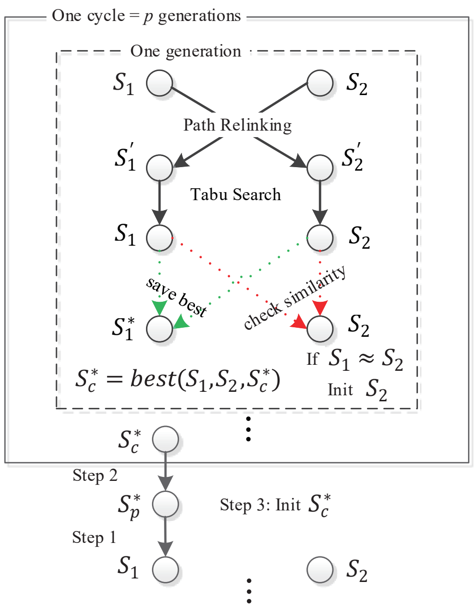

Figure 1: Diagram of MAE

Algorithm 1 MAE, a two-individual based evolutionary algorithm for FJSP

```
1:  Input: Problem instance
2:  Output: The best solution $S^*$ found
3:  $gen \leftarrow 0, S_1, S_2, S_c^*, S_p^*, S^* \leftarrow$ Init()
4:  while stopping condition is not reached do
5:      $S_1^{'} \leftarrow$ PR($S_1, S_2$), $S_2^{'} \leftarrow$ PR($S_2, S_1$)
6:      $S_1 \leftarrow$ TS($S_1^{'}$), $S_2 \leftarrow$ TS($S_2^{'}$)
7:      $S_c^* \leftarrow$ save_best($S_1, S_2, S_c^*$)
8:      $S^* \leftarrow$ save_best($S_c^*, S^*$)
9:      if $gen$ is equal to an integer parameter $p$ then
10:         $S_1 \leftarrow S_p^*$, $S_p^* \leftarrow S_c^*$, $S_c^* \leftarrow$ Init(), $gen \leftarrow 0$
11:     end if
12:     if $S_1 \approx S_2$ then
13:         $S_2 \leftarrow$ Init()
14:     end if
15:     $gen \leftarrow gen + 1$
16: end while
17: return $S^*$
```

2.2 Initial solutions and tabu search procedure

As in [González *et al.*, 2015], a solution of FJSP in MAE takes the form (α, π), where α is a feasible assignment of each operation o to a machine $M_a \in M(o)$, denoted by $\alpha(o) = a$, and π is a processing order of the operations on all machines compatible with the job sequence. At the beginning, MAE generates an initial random solution (α, π) for S_1, S_2, S_c^*, S_p^* and S^*, respectively, using the Init() function, by assigning each operation of each job to each of its candidate machines with equal probability, respecting all the constraints.

The tabu search procedure TS(S) is called in MAE to intensify the search. It improves the solution S by re-assigning a critical operation to a different machine and inserting it to a feasible position, or by changing the position of a critical operation on the same machine. In this paper, the machine re-assignment is performed on the k-insertion neighborhood (called N^k here) proposed by [Mastrolilli and Gambardella, 2000], and the position change is performed on the neighborhood called N^π and proposed by [González *et al.*, 2015]. In short, MAE repeatedly chooses the best non-tabued move from $N^\pi \cup N^k$ to perform, and the move is prohibited to be performed again within the tabu tenure, which is similar to the tabu strategy used in [Peng *et al.*, 2015].

2.3 Path relinking based recombination operator

Traditional path relinking for two individuals S_1 and S_2 consists in finding individuals $T_0, T_1, T_2, \ldots$, such that $T_0 = S_1$, and T_{i+1} is obtained by applying a single move to T_i and is closer to S_2 than T_i [Chen *et al.*, 2017]. The key issue for applying path relinking to FJSP is to define the distance between two individuals.

For example, in the scatter search for FJSP proposed by [González *et al.*, 2015], a path relinking based recombination operator is applied on two solutions S_1 and S_2 selected from a set called *RefSet*, using two distances. The first distance d^α is to measure the assignment difference, which is defined as the number of operations having a different machine assignment in S_1 and S_2, and the second distance d^π is to measure the sequence difference, which is defined as the number of pairs of operations requiring the same machine that are processed in different order. Besides, d^α has higher precedence than d^π. In order to obtain T_{i+1} from T_i, both distances d^α and d^π are considered.

In this paper, we define a unique distance between S_1 and S_2 which unifies the assignment difference and sequence difference as follows. Let M_o^S (P_o^S) denote the machine assigned to operation o (the position of o on M_o^S) in solution S, and L_a^S be the number of operations assigned to machine a in solution S. If operation o is assigned on the same machine in two solutions S_1 and S_2, we define $d_o(S_1, S_2) = |P_o^{S_1} - P_o^{S_2}|$ to be the difference of o between S_1 and S_2. Otherwise, we define $d_o(S_1, S_2) = \min\{P_o^{S_1} + P_o^{S_2}, (L^{S_1}_{M_o^{S_1}} - P_o^{S_1}) + (L^{S_2}_{M_o^{S_2}} - P_o^{S_2})\}$ to be the difference of o between S_1 and S_2. Then, the distance between S_1 and S_2 is defined as $d(S_1, S_2) = \sum_{i=1}^{n} \sum_{j=1}^{m} d_{o_{ij}}(S_1, S_2)$.

So, in order to obtain T_{i+1} from T_i, our path relinking applies a single move that changes the position of an operation on the same machine or re-assigns a different machine to an operation, such that the resulted solution is feasible and its distance to S_2 is smaller than T_i. Note that the moved operation can be non-critical. Our neighborhood is in fact $N_g^\pi \cup N_g^k$, where N_g^π (N_g^k) is extended from N^π (N^k) by including the moves of non-critical operations resulting in feasible solutions. This path relinking is much simpler thanks to the unique distance.

Algorithm 2 presents our recombination operator based on path relinking. The operator uses three parameters α, β and γ, whose value will be established empirically later. The path from the initial solution S_i to the guiding solution S_g is built step by step as follows. Let S_c be the current solution (S_c is initialized to be S_i). First, we construct the set of feasible solutions N that can be obtained from S_c by applying a single move (lines 5–11). If an operation o is on different machines in S_c and S_g, then the set $N_g^k(S_c, o)$ of feasible solutions obtained from S_c by moving operation o to the machine of o in S_g is added into N. Otherwise, the set $N_g^\pi(S_c, o)$ of feasible solutions obtained from S_c by changing the position of

Algorithm 2 A path relinking based recombination operator

```
1: Input: Initial solution $S_i$ and guiding solution $S_g$
2: Output: A reference solution $S_r$
3: $S_c \leftarrow S_i, PathSet \leftarrow \emptyset, N \leftarrow \emptyset$
4: while $d(S_c, S_g) > \alpha \times d(S_i, S_g)$ do
5:     for each operation $o$ in $S_c$ do
6:         if $M_o^{S_c} \neq M_o^{S_g}$ then
7:             $N \leftarrow N \cup N_g^k(S_c, o)$
8:         else if $M_o^{S_c} = M_o^{S_g}$ and $P_o^{S_c} \neq P_o^{S_g}$ then
9:             $N \leftarrow N \cup N_g^\pi(S_c, o)$
10:        end if
11:    end for
12:    for each solution $S \in N$ do
13:        if $d(S, S_g) > d(S_c, S_g)$ then
14:            $N \leftarrow N \setminus \{S\}$
15:        else
16:            estimate makespan $obj(S)$
17:        end if
18:    end for
19:    for each solution $S \in N$ do
20:        indexDis($S$) $\leftarrow |\{T \in N | d(T, S_g) < d(S, S_g)\}|$
21:        indexObj($S$) $\leftarrow |\{T \in N | obj(T) < obj(S)\}|$
22:    end for
23:    sort $N$ in increasing order of indexDis($S$)+indexObj($S$),
       breaking ties randomly
24:    $k \leftarrow rand\{0, 1, \ldots, \min\{\gamma, |N| - 1\}\}$
25:    $S_c \leftarrow N(k); N \leftarrow \emptyset$
26:    if $d(S_c, S_g) < \beta \times d(S_i, S_g)$ then
27:        $PathSet \leftarrow PahtSet \cup \{S_c\}$
28:    end if
29: end while
30: $S_r = \arg\min\{f(S), S \in PathSet\}$, return $S_r$
```

o on the same machine is added into N. Second, each solution of which the distance to S_g is greater than the distance of S_c to S_g is removed from N, and the makespan of each remaining solution is estimated (lines 12–16). Third, for each remaining solution S in N, the number of solutions closer to S_g (with a better makespan) is computed and is denoted by indexDis(S) (indexObj(S)) (lines 19–22). Note that indexDis(S) and indexObj(S) represent here two measures of quality for S. Fourth, we sort N in the increasing order of indexDis(S) + indexObj(S) and randomly choose one of the first γ solutions to be the next S_c along the path, and store it in $PathSet$ if its distance to S_g is smaller than $\beta \times d(S_i, S_g)$ (lines 23–28). These steps repeats until $d(S_c, S_g)$ is no longer larger than $\alpha \times d(S_i, S_g)$ (line 4). Finally, the best solution in $PathSet$ is returned as the reference solution (line 30).

3 Computational Results

3.1 Experimental protocol and benchmarks

Our MAE algorithm is implemented in C++ and runs on an Intel Xeon E5-2697 processor with 2.60 GHz CUP and 2 GB RAM. In our experiments, we set p, α, β, γ to 10, 0.4, 0.6, and 5, respectively. Two solutions are considered to be close and one of them is to be replaced with a random solution when the number of operations that have different machine assignments or different positions on the same machine is less than 10% of the total number of operations in all jobs. The maximum number of iterations of the tabu search procedure is 10000. These parameter values are determined by extensive preliminary experiments.

We evaluate the performance of MAE on four benchmarks widely used in the literature: *DPdata* [Dauzère-Pérès and Paulli, 1997], *BCdata* [Barnes and Chambers, 1998], *BRdata* [Brandimarte, 1993], and *HUdata* [Hurink *et al.*, 1994], having 178 instances in total with different sizes and flexibilities. MAE is applied on each instance with 20 independent runs. Following the common practice in the field, we use the following values to compare different methods: The average relative percentage deviation RPD of objectives over the 20 runs defined as $RPD = 100 \times (f - LB)/LB$, where f is the makespan obtained by a given algorithm, and LB is the lower bound provided in Quintiq[1]; and the average computational time $t(s)$ in seconds over the 20 runs.

In order to have a fair comparison with other algorithms, the cutoff time of MAE is set to 90 seconds for the *BRdata* and *BCdata* instances, and 5 minutes for the *DPdata* and *HUdata* instances, which is the same as that in GRASP-mELS [Kemmoé-Tchomté *et al.*, 2017]. We also provide the results of MAE with a cutoff time of 30 minutes. Besides, we normalize the computational time as the computer-independent CPU time (CI-CPU) in the same way as in [Kemmoé-Tchomté *et al.*, 2017]. Therefore, setting the speed factor of our computer as 1, the speed factor of GRASP-mELS [Kemmoé-Tchomté *et al.*, 2017], CCDS [Hmida *et al.*, 2010], SSPR [González *et al.*, 2015], HDE-N2 [Yuan and Xu, 2013a], HA [Li and Gao, 2016], and HGTS [Palacios *et al.*, 2015] are 1.09, 0.85, 0.75, 0.68, 0.50, and 0.63, respectively.

3.2 Comparison with other algorithms

We compare our MAE algorithm with the recent state-of-the-art algorithms (CDDS, HDE-N2, SSPR, HGTS, HA, and GRASP-mELS) on the four benchmark sets. The comparative results are reported in Tables 1-4. Note that columns best (avg) and t(s) are the best (average) solutions obtained and average computational time in seconds required by each algorithm, the LB values marked with $*$ denote the optimal solutions, and the best known solutions that can be obtained by each reference algorithm are indicated in bold. Rows #better, #even, and #worse give the number of instances for which the best solutions obtained by MAE within 5 minutes or 90 seconds are better, equal, and worse than each reference algorithm.

From Table 1, one observes that MAE outperforms CDDS, HDE-N2, HGTS, and HA in terms of both solution quality and computational time on the *DPdata* benchmark. Although it requires slightly more computational time than SSPR and GRASP-mELS, MAE has the least RPD values (1.01 and 1.14) for the best and average solution values. Besides, MAE obtains better results for 12 and 15 instances than SSPR and GRASP-mELS, respectively. When extending the cutoff time to 30 minutes, MAE improves the previous best known solutions for 15 instances.

From Table 2, one observes that MAE is competitive with CDDS, HGTS, and HA on the *BCdata* benchmark, because it has smaller RPD values for the best and average solution values. Besides, MAE outperforms HDE-N2 in terms of both solution quality and computational time. Compared with SSPR, MAE obtains better, equal, and worse solutions for 1, 17, and 3 instances, respectively. GRASP-mELS has better performance than MAE on the *BCdata* benchmark.

Results in Table 3 show that MAE outperforms HDE-N2, SSPR, HGTS, and GRASP-mELS in terms of both solution quality and computational time on *BRdata*. Although CCDS and HA require slightly less computational time, MAE has smaller RPD values for the best and average makespan. Besides, MAE obtains better or the same results for all the instances compared with other reference algorithms.

Table 4 presents the results of MAE in comparison with SSPR and GRASP-mELS on the *HUdata* benchmark. One observes that MAE outperforms SSPR and GRASP-mELS because MAE obtains better or equal results for all the instances with less computational time only except for GRASP-mELS on the *rdata* set. In particular, MAE improves the best results obtained by SSPR (GRASP-mELS) for 4(5), 18(19), and 13(9) instances on *edata*, *rdata*, and *vdata*, respectively.

In sum, MAE improves the previous best results obtained by SSPR and GRASP-mELS for 47 and 52 out of 178 instances on all the benchmark sets[2].

3.3 Discuss and analysis

In this subsection, we analyze our algorithm MAE. In order to show the merit of the two-individual based evolutionary framework, we compare MAE with the trajectory method called iterated tabu search (ITS) which works on a single solution. At each iteration of ITS, the tabu search procedure, which is the same as that in MAE, is performed, followed by a perturbation procedure that randomly applies $0.2 * |N_c|$ moves in $N^\pi \cup N^k$ (see Section 2.2) on the current solution or the best found solution if the number of consecutive non-improving iterations exceeds 500, where N_c is the set of critical operations.

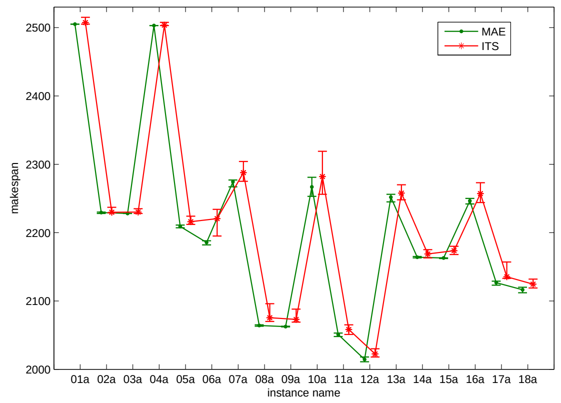

Figure 2: Comparison between MAE and ITS on *DPdata*.

Fig. 2 shows the best, average, and worst solutions obtained by MAE and ITS for each of the 18 instances in *DPdata*. It can be observed that the best, average, and worst solutions of MAE are better than or equal to those of ITS for all the instances. Besides, the differences between the best and worst solutions of MAE are also smaller than those of ITS. This indicates that the two-individual based evolutionary algorithm is superior to the trajectory method.

In order to analyze the impact of parameter p on the performance of MAE, we take 20 different values ($p \in \{1, \ldots, 20\}$), keep other parameters fixed, and apply MAE on all the instances in *DPdata*. The corresponding results are plotted in Fig. 3. One finds that the average makespan decreases when $p \in \{1, \ldots, 10\}$ and keeps flat or slightly increases when $p \in \{10, 20\}$, while the computational time

[1]http://www.quintiq.com/optimization/flexible-job-shop-scheduling-problem-results.html

[2]The detailed results can be found at www.github.com/SmartOptimization/FJSP

Table 1: Comparison between MAE and other reference algorithms on the *DPdata* instance set

Ins.	LB	2010 CCDS	2013 HDE-N2		2015 SSPR		2015 HGTS		2016 HA		2017 GRASP-mELS		This paper MAE(5 min)		This paper MAE(30 min)	
		best(avg)	best(avg)	t(s)	best(avg)	t(s)	best(avg)	t(s)	best	t(s)	best(avg)	t(s)	best(avg)	t(s)	best(avg)	t(s)
01a	2505*	2518(2525)	**2505**(2513)	838	**2505**(2508)	68	**2505**(2505)	122	2505	108	**2505**(2505)	62	**2505**(2505)	36.44	**2505**(2505)	36.44
02a	2228*	2231(2235)	2230(2231)	973	2229(2230)	100	2230(2234)	205	2230	133	2229(2231)	86	**2228**(2230.7)	145.12	**2228**(2229.9)	696.9
03a	2228*	2229(2232)	**2228**(2229)	1165	**2228**(2228)	110	**2228**(2230)	181	2229	97	**2228**(2230)	94	**2228**(2228)	48.09	**2228**(2228)	48.09
04a	2503*	**2503**(2510)	2506(2506)	850	**2503**(2504)	57	**2503**(2503)	112	2503	87	**2503**(2503)	31	**2503**(2503)	16.52	**2503**(2503)	16.52
05a	2192	2216(2218)	2212(2215)	931	2211(2215)	112	2214(2218)	208	2212	116	2212(2215)	126	2208(2211.5)	145.28	**2203**(2207.95)	784.77
06a	2163	2196(2203)	2187(2192)	1167	2183(2192)	181	2193(2198)	260	2197	93	2195(2200)	181	2182(2188.85)	171.04	**2181**(2185.6)	1019.31
07a	2216	2283(2296)	2288(2303)	1547	2274(2285)	139	2270(2280)	344	2279	204	2276(2284)	127	2269(2274.6)	167.97	**2254**(2273.75)	878.76
08a	2061*	2069(2069)	2067(2074)	1906	2064(2066)	181	2070(2074)	318	2067	184	2069(2072)	144	2063(2064.2)	135.1	**2062**(2063)	1390.7
09a	2061*	2066(2067)	2069(2073)	943	**2062**(2063)	213	2067(2069)	376	2065	201	2069(2071)	170	**2062**(2063.15)	170.11	**2062**(2063.05)	682.24
10a	2212	2291(2303)	2297(2302)	1590	2269(2287)	120	2247(2266)	369	2287	238	2263(2278)	110	2247(2266.45)	184.83	**2245**(2266.15)	1299.08
11a	2018	2063(2072)	2061(2067)	1826	2051(2058)	193	2064(2069)	294	2060	181	2065(2068)	170	2050(2051.8)	209.17	**2045**(2049.5)	1382.6
12a	1969	2031(2034)	2027(2036)	914	2018(2020)	280	2027(2033)	486	2027	151	2039(2045)	148	2016(2021.3)	207.57	**2008**(2019.65)	1126.43
13a	2197	2257(2260)	2263(2269)	2900	2248(2257)	119	2250(2264)	416	2248	293	2252(2263)	158	2247(2251.75)	137.46	**2236**(2246.3)	1488.82
14a	2161*	2167(2179)	2164(2168)	3238	2163(2164)	269	2170(2173)	396	2167	210	2170(2174)	191	2163(2163.9)	187.22	**2162**(2163)	1506.83
15a	2161*	2165(2170)	2163(2166)	2112	**2162**(2163)	376	2168(2169)	523	2163	192	2172(2174)	173	**2162**(2164.35)	193.82	**2162**(2162.9)	504.73
16a	2193	2256(2258)	2259(2266)	2802	2244(2253)	131	2246(2257)	384	2249	160	2243(2258)	151	2242(2251.7)	194.16	**2232**(2245.4)	1343.03
17a	2088	2140(2146)	2137(2141)	3096	2130(2134)	299	2142(2146)	483	2140	203	2145(2152)	190	2128(2132.7)	231.26	**2121**(2128.95)	1568.38
18a	2057	2127(2132)	2124(2128)	2489	2119(2123)	409	2129(2133)	650	2132	133	2146(2151)	164	2118(2124.8)	240.6	**2108**(2114.6)	1126.1
RPD		1.55(1.8)	1.5(1.73)		1.18(1.4)		1.34(1.59)		1.43		1.49(1.73)		1.01(1.14)		0.85(1.17)	
CI-CPU		170	1181.96		139.88		214.45		1105.03		149.94		156.76		938.87	
#better		17	16		12		14		16		15					
#even		1	2		6		4		2		3					
#worse		0	0		0		0		0		0					

Table 2: Comparison between MAE and other reference algorithms on the *BCdata* instance set

Ins.	LB	2010 CCDS	2013 HDE-N2		2015 SSPR		2015 HGTS		2016 HA		2017 GRASP-mELS		This paper MAE(90 s)		This paper MAE(30 min)	
		best(avg)	best(avg)	t(s)	best(avg)	t(s)	best(avg)	t(s)	best	t(s)	best(avg)	t(s)	best(avg)	t(s)	best(avg)	t(s)
mt10c1	927*	928(929)	**927**(928)	179	**927**(928)	26	**927**(927)	13	**927**	12	**927**(927)	8	**927**(927.35)	40.88	**927**(927)	65.38
mt10cc	908*	910(911)	**908**(911)	180	**908**(908)	20	**908**(910)	13	**908**	10	**908**(909)	17	**908**(909.9)	13.21	**908**(909.4)	109.18
mt10x	918*	**918**(918)	**918**(919)	179	**918**(918)	23	**918**(918)	15	**918**	11	**918**(918)	2	**918**(918)	33.14	**918**(918)	33.14
mt10xx	918*	**918**(918)	**918**(918)	170	**918**(918)	19	**918**(918)	12	**918**	11	**918**(918)	2	**918**(918)	6.65	**918**(918)	6.65
mt10xxx	918*	**918**(918)	**918**(918)	160	**918**(918)	20	**918**(918)	12	**918**	11	**918**(918)	2	**918**(918)	5.64	**918**(918)	5.64
mt10xy	905*	906(906)	**905**(906)	174	**905**(906)	21	**905**(905)	13	**905**	11	**905**(905)	26	**905**(905)	35.47	**905**(905)	35.47
mt10xyz	847*	849(851)	**847**(851)	166	**847**(847)	20	**847**(850)	18	**847**	9	**847**(847)	26	**847**(847.7)	32.15	**847**(847)	241.01
setb4c9	914*	919(919)	**914**(917)	338	**914**(916)	28	**914**(914)	16	**914**	15	**914**(914)	11	**914**(918.25)	43.5	**914**(914)	326.14
setb4cc	907*	909(911)	**907**(910)	336	**907**(907)	21	**907**(908)	15	**907**	15	**907**(907)	29	**907**(907)	17.39	**907**(907)	17.39
setb4x	925*	**925**(925)	**925**(926)	354	**925**(925)	19	**925**(925)	15	**925**	13	**925**(925)	4	**925**(925)	14.7	**925**(925)	14.7
setb4xx	925*	**925**(925)	**925**(926)	330	**925**(925)	21	**925**(925)	14	**925**	5	**925**(925)	2	**925**(925)	7.59	**925**(925)	7.59
setb4xxx	925*	**925**(925)	**925**(926)	315	**925**(925)	22	**925**(925)	15	**925**	9	**925**(925)	3	**925**(925)	7.22	**925**(925)	7.22
setb4xy	910*	916(916)	**910**(914)	313	**910**(912)	32	**910**(910)	19	**910**	12	**910**(910)	18	**910**(910)	43.39	**910**(910)	43.39
setb4xyz	902*	905(907)	903(905)	317	905(905)	21	905(905)	15	905	14	**902**(904)	11	**902**(905.6)	36.9	**902**(903.75)	411.96
seti5c12	1169*	1174(1175)	1171(1175)	1113	1170(1173)	25	1170(1171)	41	1170	31	**1169**(1172)	39	1170(1174.35)	41.85	1170(1173.45)	216.16
seti5cc	1135*	1136(1137)	1136(1138)	1079	**1135**(1136)	29	1136(1137)	34	1136	17	**1135**(1136)	24	1136(1136)	34.23	**1135**(1135.65)	243.58
seti5x	1198*	1201(1202)	1200(1206)	1087	**1198**(1199)	41	1199(1201)	38	**1198**	27	**1198**(1199)	36	**1198**(1201.5)	70.86	**1198**(1199.4)	460.91
seti5xx	1194*	1199(1199)	1197(1203)	1251	1197(1199)	37	1197(1198)	34	1197	29	**1194**(1197)	26	1197(1198.5)	39.47	1197(1197)	531.53
seti5xxx	1194*	1197(1198)	1197(1202)	1244	**1194**(1198)	38	1197(1198)	31	1197	19	**1194**(1197)	27	1197(1198.65)	46.37	**1194**(1196.65)	498.73
seti5xy	1135*	1136(1138)	1136(1138)	1141	**1135**(1136)	29	1136(1137)	34	1136	17	**1135**(1136)	28	1136(1136.1)	24.59	**1135**(1136)	285.01
seti5xyz	1125*	**1125**(1125)	**1125**(1130)	1223	**1125**(1126)	35	**1125**(1126)	43	**1125**	33	**1125**(1127)	42	**1125**(1128.7)	29.34	**1125**(1125.6)	268.12
RPD		0.19(0.26)	0.05(0.3)		0.03(0.12)		0.07(0.13)		0.05		0(0.07)		0.04(0.17)		0.02(0.08)	
CI-CPU		12.75	377.2		19.54		13.8		7.88		19.88		29.74		182.33	
#better		11	3		1		2		1		0					
#even		10	18		17		19		20		16					
#worse		0	0		3		0		0		5					

Table 3: Comparison between MAE and other reference algorithms on the *BRdata* instance set

Ins.	LB	2010 CCDS	2013 HDE-N2		2015 SSPR		2015 HGTS		2016 HA		2017 GRASP-mELS		This paper MAE(90 s)		This paper MAE(30 min)	
		best(avg)	best(avg)	t(s)	best(avg)	t(s)	best(avg)	t(s)	best	t(s)	best(avg)	t(s)	best(avg)	t(s)	best(avg)	t(s)
Mk01	40*	**40**(40	**40**(40)	4	**40**(40)	11	**40**(40)	5	**40**	0	**40**(40)	0	**40**(40)	0.2	**40**(40)	0.2
Mk02	26*	**26**(26	**26**(26)	6	**26**(26)	15	**26**(26)	15	**26**	1	**26**(26)	10	**26**(26)	0.52	**26**(26)	0.52
Mk03	204*	**204**(204	**204**(204)	31	**204**(204)	24	**204**(204)	2	**204**	0	**204**(204)	0	**204**(204)	0.18	**204**(204)	0.18
Mk04	60*	**60**(60	**60**(60)	13	**60**(60)	19	**60**(60)	10	**60**	0	**60**(60)	0	**60**(60)	0.51	**60**(60)	0.51
Mk05	172*	173(174	**172**(173)	38	**172**(172)	57	**172**(172)	18	**172**	5	**172**(173)	15	**172**(172)	1.46	**172**(172)	1.46
Mk06	57*	58(59	**57**(59)	98	**57**(58)	40	**57**(58)	63	**57**	54	58(58)	36	**57**(58.05)	33.65	**57**(57.4)	204.39
Mk07	139*	**139**(139	**139**(139)	26	**139**(141)	84	**139**(139)	33	**139**	20	**139**(140)	32	**139**(139.7)	53.5	**139**(139)	432.9
Mk08	523*	**523**(523	**523**(523)	189	**523**(523)	83	**523**(523)	3	**523**	0	**523**(523)	0	**523**(523)	0.45	**523**(523)	0.45
Mk09	307*	**307**(307	**307**(307)	123	**307**(307)	52	**307**(307)	24	**307**	1	**307**(307)	0	**307**(307)	1.47	**307**(307)	1.47
Mk10	189	197(198)	198(202)	266	196(197)	94	198(199)	104	197	33	197(199)	59	195(195.95)	40.21	**193**(194.6)	763.92
RPD		0.66(0.94)	0.48(1.1)		0.37(0.74)		0.48(0.71)		0.42		0.6(0.83)		0.35(0.43)		0.23(0.34)	
CI-CPU		12.75	53.99		35.93		17.45		5.7		16.57		13.22		140.6	
#better		3	1		1		1		1		2					
#even		7	9		9		9		9		8					
#worse		0	0		0		0		0		0					

Table 4: Comparison between MAE and other reference algorithms on the *HUdata* instance set

Ins.	*edata*						*rdata*						*vdata*					
	GRASP-mELS		SSPR		MAE(5 min)		GRASP-mELS		SSPR		MAE(5 min)		GRASP-mELS		SSPR		MAE(5 min)	
	RPD$_{best}$	RPD$_{avg}$	RPD$_{best}$	RPD$_{avg}$	RPD$_{best}$	RPD$_{avg}$	RPD$_{best}$	RPD$_{avg}$	RPD$_{best}$	RPD$_{avg}$	RPD$_{best}$	RPD$_{avg}$	RPD$_{best}$	RPD$_{avg}$	RPD$_{best}$	RPD$_{avg}$	RPD$_{best}$	RPD$_{avg}$
mt06/10/20	0	0	0	0.04	0	0.07	0	0	0	0	0	0	0	0	0	0	0	0
la01-la05	0	0	0	0	0	0	0	0.07	0.07	0.09	0	0.07	0	0	0	0	0	0
la06-la10	0	0	0	0	0	0	0	0	0	0.01	0	0	0	0	0	0	0	0
la11-la15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
la16-la20	0	0	0	0	0	0	0	0	0	0.03	0	0	0	0	0	0	0	0
la21-la25	0	0.24	0.08	0.23	0	0.22	2.63	3.27	2.53	2.91	1.91	2.38	0.49	0.8	0.23	0.35	0.1	0.24
la26-la30	0.33	0.61	0.43	0.66	0.3	0.71	0.36	0.71	0.36	0.48	0.13	0.27	0.17	0.24	0.06	0.08	0	0.03
la31-la35	0.05	0.11	0.01	0.07	0	0.01	0.05	0.12	0.04	0.05	0	0.02	0.04	0.07	0.01	0.02	0	0
la36-la40	0	0.04	0	0.05	0	0.02	0.36	1.22	0.66	0.9	0	0.1	0	0	0	0	0	0
CI-CPU	22.98		28.26		19.17		51.23		34.78		41.95		30.25		47.83		26.34	
#better	4		5				18		19				13		9			
#even	39		38				25		24				30		34			
#worse	0		0				0		0				0		0			

drastically decreases when $p \in \{1, 2, 3\}$ and gradually increases when $p \in \{3, \ldots, 20\}$. The reason might lie in the fact that when p is small, the best solution preserved in the previous cycle is not of high quality, which cannot provide good features to be inherited. When p is too large, the best solution is closer to the best solution found so far, which cannot provide sufficient diversity, so that MAE would be more likely trapped into local optima. The best value of p in MAE is suggested to be 10.

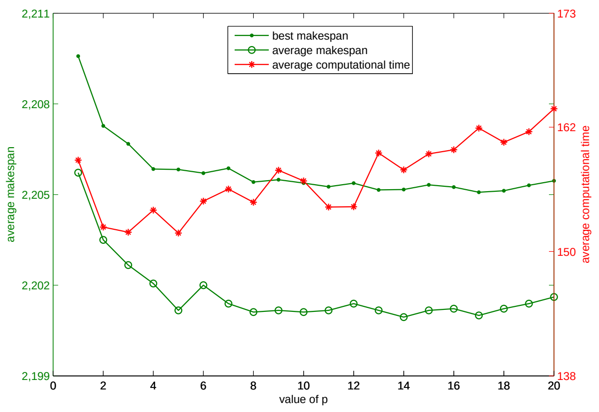

Figure 3: The average makespan and computational time corresponding to different values of parameter p on *DPdata*.

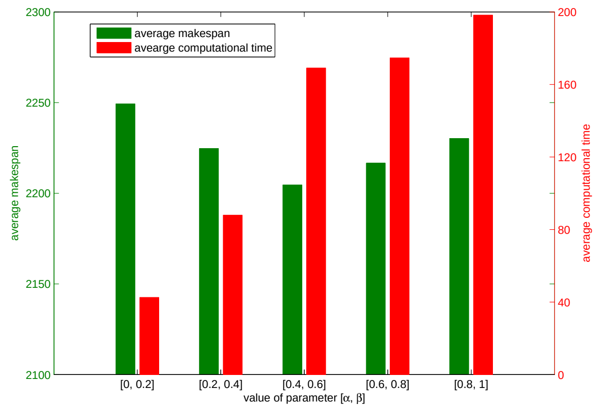

Figure 4: The average makespan and computational time corresponding to different values of parameter $[\alpha, \beta]$ on *DPdata*.

In order to analyze the impact of the parameter α, β, γ on MAE, we take 5 groups of values $([0, 0.2], \ldots, [0.8, 1])$ for $[\alpha, \beta]$, 15 values $(1, \ldots, 15)$ for γ, keep other parameters fixed, and conduct experiments on all the instances in *DPdata* and *BCdata*, respectively. The corresponding results are

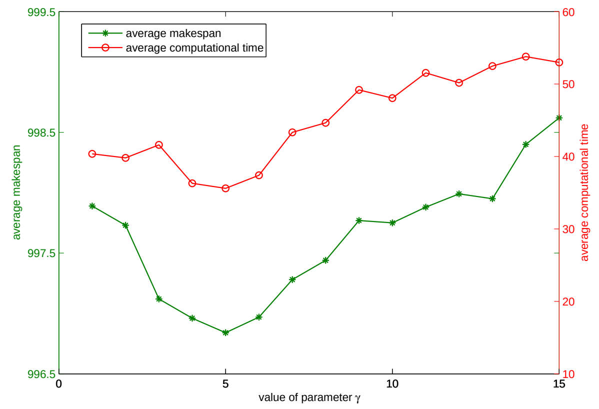

Figure 5: The average makespan and computational time corresponding to different values of parameter γ on *BCdata*.

presented in Fig. 4 and Fig. 5. From Fig. 4, one observes that the average makespan decreases from the first to the third group and increases from the third to the fifth group, while the corresponding computational time gradually increases in the whole range. From Fig. 5, one observes that both average makespan and computational time decrease when $\gamma \in [1, 5]$ and increase when $\gamma \in [5, 15]$. Considering both solution quality and computational efficiency, α, β, γ are suggested to be 0.4, 0.6, and 5, respectively.

4 Conclusion

We have proposed a master-apprentice evolutionary algorithm called MAE for solving the flexible job shop scheduling problem, which distinguishes itself from both single solution-based and traditional population-based metaheuristics in three main aspects: (1) The population size in MAE is two, allowing effective collaboration between the two individuals; (2) MAE uses a simple but very effective individual updating strategy to ensure the quality and the diversity of the evolution; (3) In order to generate promising offspring solutions, MAE uses a semantic problem-specific recombination operator based on path relinking with a novel distance definition for two individuals. Computational experiments show the high performance of MAE in terms of both solution quality and computational efficiency by improving the previous best known results for 47 instances while matching the best known results for others except for only 5 instances. We strongly believe that in spite of its simplicity, this two-individual based mater-apprentice evolutionary algorithm is a promising framework for solving other challenging combinatorial optimization problems.

References

[Ansotegui *et al.*, 2015] Carlos Ansotegui, Yuri Malitsky, Horst Samulowitz, Meinolf Sellmann, and Kevin Tierney. Model-based genetic algorithms for algorithm configuration. In *International Joint Conference on Artificial Intelligence*, pages 1854–1861, 2015.

[Barnes and Chambers, 1998] John Wesley Barnes and John Burges Chambers. Flexible job shop scheduling by tabu search. Technical report, The University of Texas at Austin, June 1998.

[Bożejko *et al.*, 2010] Wojciech Bożejko, Mariusz Uchroński, and Mieczysław Wodecki. Parallel hybrid metaheuristics for the flexible job shop problem. *Computers and Industrial Engineering*, 59(2):323–333, 2010.

[Brandimarte, 1993] Paolo Brandimarte. Routing and scheduling in a flexible job shop by tabu search. *Annals of Operations Research*, 41(3):157–183, 1993.

[Brucker and Schlie, 1991] Peter Brucker and Rainer Schlie. *Job-shop scheduling with multi-purpose machines*. Springer-Verlag New York, Inc., 1991.

[Chen *et al.*, 2017] Jingwei Chen, Robert C. Holte, Sandra Zilles, and Nathan R. Sturtevant. Front-to-end bidirectional heuristic search with near-optimal node expansions. In *International Joint Conference on Artificial Intelligence*, pages 2033–2040, 2017.

[Dauzère-Pérès and Paulli, 1997] Stéphane Dauzère-Pérès and Jan Paulli. An integrated approach for modeling and solving the general multiprocessor job-shop scheduling problem using tabu search. *Annals of Operations Research*, 70(1):281–306, 1997.

[Gao *et al.*, 2008] Jie Gao, Linyan Sun, and Mitsuo Gen. A hybrid genetic and variable neighborhood descent algorithm for flexible job shop scheduling problems. *Computers & Operations Research*, 35(9):2892–2907, 2008.

[Gao *et al.*, 2016] Kai-Zhou Gao, Ponnuthurai Nagaratnam Suganthan, Quan-Ke Pan, Tay Jin Chua, Tian-Xiang Cai, and Chin-Soon Chong. Discrete harmony search algorithm for flexible job shop scheduling problem with multiple objectives. *Journal of Intelligent Manufacturing*, 27(2):363–374, 2016.

[Garey *et al.*, 1976] Michael Randolph Garey, David Stifler Johnson, and Ravi Sethi. The complexity of flowshop and jobshop scheduling. *Mathematics of Operations Research*, 1(2):117–129, 1976.

[Gomes *et al.*, 2013] Marta Castilho Gomes, Ana Paula Barbosa-Póvoa, and Augusto Queiroz Novais. Reactive scheduling in a make-to-order flexible job shop with re-entrant process and assembly: a mathematical programming approach. *International Journal of Production Research*, 51(17):5120–5141, 2013.

[González *et al.*, 2015] Miguel Ángel. González, Camino Rodríguez Vela, and Ramiro Varela. Scatter search with path relinking for the flexible job shop scheduling problem. *European Journal of Operational Research*, 245(1):35–45, 2015.

[Gutiérrez and García-Magariño, 2011] Celia Gutiérrez and Iván García-Magariño. Modular design of a hybrid genetic algorithm for a flexible job–shop scheduling problem. *Knowledge-Based Systems*, 24(1):102–112, 2011.

[Hansmann *et al.*, 2014] Ronny S. Hansmann, Thomas Rieger, and Uwe T. Zimmermann. Flexible job shop scheduling with blockages. *Mathematical Methods of Operations Research*, 79(2):135–161, 2014.

[Hmida *et al.*, 2010] Abir Ben Hmida, Mohamed Haouari, and Pierre Lopez. Discrepancy search for the flexible job shop scheduling problem. *Computers & Operations Research*, 37(12):2192–2201, 2010.

[Hurink *et al.*, 1994] Johann Hurink, Bernd Jurisch, and Monika Thole. Tabu search for the job-shop scheduling problem with multi-purpose machines. *Operations-Research-Spektrum*, 15(4):205–215, 1994.

[Kemmoé-Tchomté *et al.*, 2017] Sylverin Kemmoé-Tchomté, Damien Lamy, and Nikolay Tchernev. An effective multi-start multi-level evolutionary local search for the flexible job-shop problem. *Engineering Applications of Artificial Intelligence*, 62:80–95, 2017.

[Lau *et al.*, 2013] Hoong Chuin Lau, Lucas Agussurja, Shih Fen Cheng, and Pang Jin Tan. A multi-objective memetic algorithm for vehicle resource allocation in sustainable transportation planning. In *International Joint Conference on Artificial Intelligence*, pages 2833–2839, 2013.

[Li and Gao, 2016] Xinyu Li and Liang Gao. An effective hybrid genetic algorithm and tabu search for flexible job shop scheduling problem. *International Journal of Production Economics*, 174:93–110, 2016.

[Li *et al.*, 2010] Jun-Qing Li, Quan-Ke Pan, and Yun-Chia Liang. An effective hybrid tabu search algorithm for multi-objective flexible job-shop scheduling problems. *Computers & Industrial Engineering*, 59(4):647 – 662, 2010.

[Lü *et al.*, 2010] Zhipeng Lü, Fred Glover, and Jin Kao Hao. A hybrid metaheuristic approach to solving the ubqp problem. *European Journal of Operational Research*, 207(3):1254–1262, 2010.

[Mastrolilli and Gambardella, 2000] Monaldo Mastrolilli and Luca Maria Gambardella. Effective neighborhood functions for the flexible job shop problem. *Journal of Scheduling*, 3(1):3–20, 2000.

[Moalic and Gondran, 2017] Laurent Moalic and Alexandre Gondran. Variations on memetic algorithms for graph coloring problems. *Journal of Heuristics*, pages 1–24, 2017.

[Ángel González *et al.*, 2013] Miguel Ángel González, Camino Rodríguez Vela, and Ramiro Varela. An efficient memetic algorithm for the flexible job shop with setup times. In *International Conference on International Conference on Automated Planning and Scheduling*, pages 91–99, 2013.

[Oddi *et al.*, 2011] Angelo Oddi, Riccardo Rasconi, Amedeo Cesta, and Stephen F. Smith. Iterative flattening search for the flexible job shop scheduling problem. In *International Joint Conference on Artificial Intelligence*, pages 1991–1996, 2011.

[Özgüven *et al.*, 2010] Cemal Özgüven, Lale Özbakır, and Yasemin Yavuz. Mathematical models for job-shop scheduling problems with routing and process plan flexibility. *Applied Mathematical Modelling*, 34(6):1539–1548, 2010.

[Palacios *et al.*, 2015] Juan José Palacios, Miguel Ángel González, CaminoRodríguez Vela, Inés González-Rodríguez, and Jorge Puente. Genetic tabu search for the fuzzy flexible job shop problem. *Computers & Operations Research*, 54(C):74–89, 2015.

[Peng *et al.*, 2015] Bo Peng, Zhipeng Lü, and T. C. E. Cheng. A tabu search/path relinking algorithm to solve the job shop scheduling problem. *Computers & Operations Research*, 53(53):154–164, 2015.

[Pezzella *et al.*, 2008] Ferdinando Pezzella, Gianluca Morganti, and Gianfranco Ciaschetti. A genetic algorithm for the flexible job-shop scheduling problem. *Computers & Operations Research*, 35(10):3202–3212, 2008.

[Roshanaei *et al.*, 2013] Vahid Roshanaei, Ahmed Azab, and Hoda Elmaraghy. Mathematical modelling and a meta-heuristic for flexible job shop scheduling. *International Journal of Production Research*, 51(20):6247–6274, 2013.

[Thomalla, 2005] Christoph Thomalla. Job shop scheduling with alternative process plans. *International Journal of Production Economics*, 74(1):125–134, 2005.

[Wang *et al.*, 2012] Ling Wang, Gang Zhou, Ye Xu, Shengyao Wang, and Min Liu. An effective artificial bee colony algorithm for the flexible job-shop scheduling problem. *International Journal of Advanced Manufacturing Technology*, 56(60):1–8, 2012.

[Wang *et al.*, 2013] Ling Wang, Gang Zhou, Ye Xu, and Min Liu. A hybrid artificial bee colony algorithm for the fuzzy flexible job-shop scheduling problem. *International Journal of Production Research*, 51(12):3593–3608, 2013.

[Yu *et al.*, 2013] Yang Yu, Xin Yao, and Zhi-Hua Zhou. On the approximation ability of evolutionary optimization with application to minimum set cover. In *Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence*, pages 3190–3194. AAAI Press, 2013.

[Yuan and Xu, 2013a] Yuan Yuan and Hua Xu. Flexible job shop scheduling using hybrid differential evolution algorithms. *Computers & Industrial Engineering*, 65(2):246–260, 2013.

[Yuan and Xu, 2013b] Yuan Yuan and Hua Xu. An integrated search heuristic for large-scale flexible job shop scheduling problems. *Computers & Operations Research*, 40(12):2864–2877, 2013.