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

Junwen Ding[1], **Zhipeng Lü**[1,*], **Chu-Min Li**[2], **Liji Shen**[3], **Liping Xu**[1], **Fred Glover**[4]
[1]School of Computer Science, Huazhong University of Science and Technology, Wuhan, China
[2]MIS, University of Picardie Jules Verne, Amiens, France
[3]Operations Management, WHU-Otto Beisheim School of Management, Vallendar, Germany
[4]College of Engineering & Applied Science, University of Colorado, Colorado, USA
{dingjunwen,zhipeng.lv,xlphust}@hust.edu.cn, chu-min.li@u-picardie.fr, liji.shen@whu.edu, glover@colorado.edu

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). 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 313 widely-used public instances show that MAE improves the previous best known results for 47 instances and matches the best known results on all except 3 of the remaining instances while consuming the same computational time as current state-of-the-art metaheuristics. MAE additionally establishes solution quality records for 10 hard instances whose previous best values were established by a well-known industrial solver and a state-of-the-art exact method.

Introduction

The job shop scheduling problem (JSP) is a strongly NP-hard problem (Garey, Johnson, and Sethi 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 maximum 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, Özbakr, and Yavuz 2010) with routing and process plan flexibility, and a mixed-integer linear optimization model combined with a branch and bound algorithm proposed by (Hansmann, Rieger, and Zimmermann 2014). Other exact methods based on mixed-integer linear programming can be found in (Gomes, Barbosa-Pvoa, and Novais 2013; Roshanaei, Azab, and Elmaraghy 2013).

For large FJSP instances, various metaheuristic algorithms have been employed. The noteworthy literatures include: (Brandimarte 1993), (Dauzère-Pérès and Paulli 1997), (Mastrolilli and Gambardella 2000), (Pezzella, Morganti, and Ciaschetti 2008), (Gao, Sun, and Gen 2008), (Oddi et al. 2011), (Bożejko, Uchroński, and Wodecki 2010), (Gutiérrez and García-Magariño 2011), (Li, Pan, and Liang 2010), (Wang et al. 2012), (Wang et al. 2013), (Yuan and Xu 2013a), (González, Vela, and Varela 2013), and (Gao et al. 2016). Recent approaches include the climbing depth-bounded discrepancy search (CDDS) algorithm (Hmida, Haouari, and Lopez 2010), hybrid differential evolution algorithm (HDE-N2) (Yuan and Xu 2013b), scatter search with path relinking (SSPR) (González, Vela, and Varela 2015), genetic tabu search (HGTS) (Palacios et al. 2015), hybrid genetic algorithm with tabu search (HA) (Li and Gao 2016), and multi-start multi-level evolutionary local search (GRASP-mELS) (Kemmoé-Tchomté, Lamy, and Tchernev 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 (Lahiri and Cebrian 2010). 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 par-

*Corresponding author

ticularity 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. Similar attempts of two-individual memetic algorithm hybridized with regular re-initialization can be found in (Duarte et al. 2005).

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.

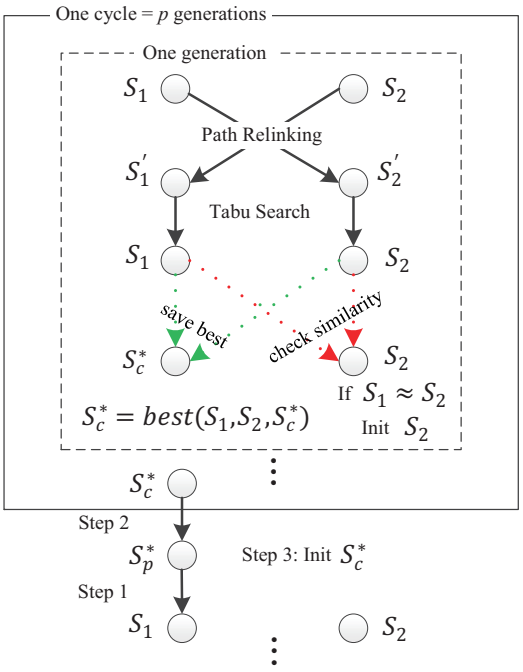

Figure 1: Diagram of MAE.

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.

Two-individual based evolution mechanism is a unique feature of MAE. Traditional population-based evolutionary algorithms usually confront with the drawback of maintaining large population and high consumption of computing resources. 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.

Main scheme of MAE

MAE follows the basic framework of the evolutionary algorithms (Lü, Glover, and Hao 2010; Sutton and Neumann 2012; Yu, Yao, and Zhou 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.

Initial solutions and tabu search procedure

As in (González, Vela, and Varela 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 random solutions 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. Note that the operations in the critical path are called critical operations, and critical

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: $S_1^{'} \leftarrow \text{PR}(S_1, S_2)$, $S_2^{'} \leftarrow \text{PR}(S_2, S_1)$
6: $S_1 \leftarrow \text{TS}(S_1^{'})$, $S_2 \leftarrow \text{TS}(S_2^{'})$
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^*

path is the longest path in the disjunctive graph representation of a schedule. 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, Vela, and Varela 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, Lü, and Cheng 2015).

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. 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, Vela, and Varela 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

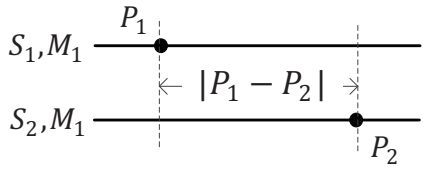

Figure 2: The distance of the operation on the same machine.

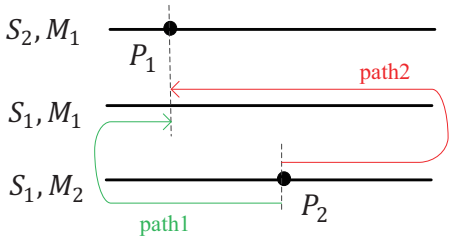

Figure 3: The distance of the operation on the different machine.

$d_o(S_1, S_2) = |P_o^{S_1} - P_o^{S_2}|$ to be the difference of o between S_1 and S_2 (see Fig. 2). Otherwise, o can be moved to the beginning (end) of $M_o^{S_1}$, and then from the beginning (end) of $M_o^{S_2}$ of S_1 to the same position as in $M_o^{S_2}$ of S_2 (see path1 (path2) in Fig. 3). The minimum one between path1 and path2 is chosen as the difference of o between S_1 and S_2, which is $d_o(S_1, S_2) = \min\{P_o^{S_1} + P_o^{S_2}, (L_{M_o^{S_1}}^{S_1} - P_o^{S_1}) + (L_{M_o^{S_2}}^{S_2} - P_o^{S_2})\}$. Then, the distance between S_1 and S_2 is defined as $d(S_1, S_2) = \sum_{i=1}^{n} \sum_{j=1}^{m} d_{o_{ij}}(S_1, S_2)$.

Therefore, 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 closer to S_2 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 5–13). For each operation o, if o is on different machines in S_c and S_g, let $N_g^k(S_c, o)$ be the set of feasible solutions obtained from S_c by moving o to the machine of o in S_g. Otherwise, let $N_g^\pi(S_c, o)$ be the set of feasible solutions obtained from S_c by changing the position of o on the same machine. Let S_{min} be the solution such that $d_o(S_{min}, S_g)$ is minimum (ties are broken randomly). Then, S_{min} in $N_g^k(S_c, o)$ or $N_g^\pi(S_c, o)$ is added into N. Second, each solution S such that $d(S, S_g) > d(S_c, S_g)$

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

is removed from N, and the makespan of each remaining solution is estimated (lines 14–18). 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 21–24). 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 25–30). 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 32).

It is obvious that the maximum size of set N is n_o where n_o is the number of all the operations in all the jobs, i.e., $n_o = \sum_{j=1}^{n} n_j$. The worst time complexity of lines 5-13 is $O(n_o^2)$. Lines 21-25 are actually sorting the solutions of set N, where the time complexity is $O(|N| \log |N|)$. Therefore, the worst time complexity of Algorithm 2 is $O(n_o^3)$.

Computational Results

Experimental protocol and benchmarks

Our MAE algorithm is implemented in C++ and runs on an Intel Xeon E5-2697 processor with 2.60 GHz CPU 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, Jurisch, and Thole 1994), having 313 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. We also provide the results of MAE with a cutoff time of 1 hour. Besides, we normalize the computational time as the computer-independent CPU time (CI-CPU) in the same way as that in GRASP-mELS. Therefore, setting the speed factor of our computer as 1, the speed factor of GRASP-mELS, CCDS, SSPR, HDE-N2, HA, and HGTS are 1.09, 0.85, 0.75, 0.68, 0.50, and 0.63, respectively.

Comparison with metaheuristics

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

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

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(1 hour)	
		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)	28.56	**2505**(2505)	28.56
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)	712.33
03a	2228*	2229(2232)	**2228**(2229)	1165	**2228**(2228)	110	**2228**(2230)	181	2229	97	**2228**(2230)	94	**2228**(2228)	55.8	**2228**(2228)	55.8
04a	2503*	**2503**(2510)	2506(2506)	850	**2503**(2504)	57	**2503**(2503)	112	2503	87	**2503**(2503)	31	**2503**(2503)	8.62	**2503**(2503)	8.62
05a	2192	2216(2218)	2212(2215)	931	2211(2215)	112	2214(2218)	208	2212	116	2212(2215)	126	2208(2211.45)	125.69	**2203**(2208.05)	834.49
06a	2163	2196(2203)	2187(2192)	1167	2183(2192)	181	2193(2198)	260	2197	93	2195(2200)	181	2182(2188.85)	177.42	**2181**(2184.3)	1867.14
07a	2216	2283(2296)	2288(2303)	1547	2274(2285)	139	2270(2280)	344	2279	204	2276(2284)	127	2269(2274.6)	180.3	**2254**(2273.85)	2316.78
08a	2061*	2069(2069)	2067(2074)	1906	2064(2066)	181	2070(2074)	318	2067	184	2069(2072)	144	2063(2064.3)	122.58	**2062**(2063.4)	1741.55
09a	2061*	2066(2067)	2069(2073)	943	**2062**(2063)	213	2067(2069)	376	2065	201	2069(2071)	170	**2062**(2063.15)	176.44	**2062**(2063.1)	472.62
10a	2212	2291(2303)	2297(2302)	1590	2269(2287)	120	2247(2266)	369	2287	238	2263(2278)	110	2247(2266.4)	224.36	**2245**(2266.15)	2428.7
11a	2018	2063(2072)	2061(2067)	1826	2051(2058)	193	2064(2069)	294	2060	181	2065(2068)	170	2050(2051.8)	200.57	**2045**(2049.75)	2865.49
12a	1969	2031(2034)	2027(2036)	914	2018(2020)	280	2027(2033)	486	2027	151	2039(2045)	148	2016(2021.45)	215.64	**2008**(2019.3)	1588.16
13a	2197	2257(2260)	2263(2269)	2900	2248(2257)	119	2250(2264)	416	2248	293	2252(2263)	158	2247(2251.75)	116.55	**2236**(2246.65)	2674.32
14a	2161*	2167(2179)	2164(2168)	3238	2163(2164)	269	2170(2173)	396	2167	210	2170(2174)	191	2163(2163.9)	191.26	**2162**(2163.2)	2915.81
15a	2161*	2165(2170)	2163(2166)	2112	**2162**(2163)	376	2168(2169)	523	2163	192	2172(2174)	173	**2162**(2164.35)	203.2	**2162**(2163.15)	568.14
16a	2193	2256(2258)	2259(2266)	2802	2244(2253)	131	2246(2257)	384	2249	160	2243(2258)	151	2242(2251.65)	196.5	**2232**(2245.45)	2135.66
17a	2088	2140(2146)	2137(2141)	3096	2130(2134)	299	2142(2146)	483	2140	203	2145(2152)	190	2128(2132.7)	245.71	**2121**(2129.3)	1682.54
18a	2057	2127(2132)	2124(2128)	2489	2119(2123)	409	2129(2133)	650	2132	133	2146(2151)	164	2118(2124.85)	242.2	**2108**(2114.6)	1752.68
RPD		1.55(1.8)	1.5(1.73)		1.18(1.4)		1.34(1.59)		1.43		1.49(1.73)		1.04(1.24)		0.85(1.13)	
CI-CPU		170	1181.96		139.88		214.45		1105.03		149.94		158.7		1480.52	
#better		17	16		12		14		16		15					
#even		1	2		6		4		2		3					
#worse		0	0		0		0		0		0					

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 (1.04 and 1.24) for the best and average objective values. Besides, MAE obtains better results for 12 and 15 instances than SSPR and GRASP-mELS, respectively. When extending the cutoff time to 1 hour, 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 objective 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, 19, and 1 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 GRASP-mELS(SSPR) 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 the 178 test instances.

Comparison with the state-of-the-art exact method

We compare MAE with the state-of-the-art exact method based on constraint programming: LNS+FDS (Vilím, Laborie, and Shaw 2015). LNS+FDS constitutes the basis of the automatic search for scheduling problems in CP Optimizer, which is part of IBM ILOG CPLEX Optimization Studio. LNS+FDS (also denoted as CPO) has been successfully tested on a range of scheduling benchmarks such as JSP and FJSP, etc.

MAE obtains better, equal, and worse results for 45, 255, and 13 instances compared with CPO among the 313 instances, respectively. Table 5 reports the results of MAE and comparison with CPO for the 45 improving instances. Note that the results of CPO were obtained with a time limit of 8 hours, while the results of MAE were obtained with a time limit of 1 hour.

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(1 hour)	
		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**(927.3)	45.72	**927**(927)	61.69
mt10cc	908*	910(911)	**908**(911)	180	**908**(908)	20	**908**(910)	13	**908**	10	**908**(909)	17	**908**(909.85)	14.25	**908**(909.4)	125.5
mt10x	918*	**918**(918)	**918**(919)	179	**918**(918)	23	**918**(918)	15	**918**	11	**918**(918)	2	**918**(918)	25.67	**918**(918)	25.67
mt10xx	918*	**918**(918)	**918**(918)	170	**918**(918)	19	**918**(918)	12	**918**	11	**918**(918)	2	**918**(918)	4.5	**918**(918)	4.5
mt10xxx	918*	**918**(918)	**918**(918)	160	**918**(918)	20	**918**(918)	12	**918**	11	**918**(918)	2	**918**(918)	6.78	**918**(918)	6.78
mt10xy	905*	906(906)	**905**(906)	174	**905**(906)	21	**905**(905)	13	**905**	11	**905**(905)	26	**905**(905)	34.42	**905**(905)	34.42
mt10xyz	847*	849(851)	**847**(851)	166	**847**(847)	20	**847**(850)	18	**847**	9	**847**(847)	26	**847**(847.65)	35.46	**847**(847)	256.38
setb4c9	914*	919(919)	**914**(917)	338	**914**(916)	28	**914**(914)	16	**914**	15	**914**(914)	11	**914**(918.25)	39.78	**914**(914)	302.61
setb4cc	907*	909(911)	**907**(910)	336	**907**(907)	21	**907**(908)	15	**907**	15	**907**(907)	29	**907**(907)	12.54	**907**(907)	12.54
setb4x	925*	**925**(925)	**925**(926)	354	**925**(925)	19	**925**(925)	15	**925**	13	**925**(925)	4	**925**(925)	16.42	**925**(925)	16.42
setb4xx	925*	**925**(925)	**925**(926)	330	**925**(925)	21	**925**(925)	14	**925**	5	**925**(925)	2	**925**(925)	7.7	**925**(925)	7.7
setb4xxx	925*	**925**(925)	**925**(926)	315	**925**(925)	22	**925**(925)	15	**925**	9	**925**(925)	3	**925**(925)	8.45	**925**(925)	8.45
setb4xy	910*	916(916)	**910**(914)	313	**910**(912)	32	**910**(910)	19	**910**	12	**910**(910)	18	**910**(910)	58.79	**910**(910)	58.79
setb4xyz	902*	905(907)	903(905)	317	905(905)	21	905(905)	15	905	14	**902**(904)	11	**902**(905.6)	34.6	**902**(903.55)	956.86
seti5c12	1169*	1174(1175)	1171(1175)	1113	1170(1173)	25	1170(1171)	41	1170	31	**1169**(1172)	39	1170(1174.4)	64.13	1170(1173.2)	205.68
seti5cc	1135*	1136(1137)	1136(1138)	1079	**1135**(1136)	29	1136(1137)	34	1136	17	**1135**(1136)	24	**1135**(1136.2)	32.41	**1135**(1135.65)	243.52
seti5x	1198*	1201(1202)	1200(1206)	1087	**1198**(1199)	41	1199(1201)	38	**1198**	27	**1198**(1199)	36	**1198**(1201.6)	75.48	**1198**(1199.35)	341.4
seti5xx	1194*	1199(1199)	1197(1203)	1251	1197(1199)	37	1197(1198)	34	1197	29	**1194**(1197)	26	1197(1198.5)	45.76	1197(1197)	473.48
seti5xxx	1194*	1197(1198)	1197(1202)	1244	**1194**(1198)	38	1197(1198)	31	1197	19	**1194**(1197)	27	1197(1198.45)	35.5	**1194**(1196.7)	612.57
seti5xy	1135*	1136(1138)	1136(1138)	1141	**1135**(1136)	29	1136(1137)	34	1136	17	**1135**(1136)	28	**1135**(1136.4)	25.53	**1135**(1136)	227.91
seti5xyz	1125*	**1125**(1125)	**1125**(1130)	1223	**1125**(1126)	35	**1125**(1126)	43	**1125**	33	**1125**(1127)	42	**1125**(1128.75)	32.96	**1125**(1125.65)	336.27
RPD		0.19(0.26)	0.05(0.3)		0.03(0.12)		0.07(0.13)		0.05		0(0.07)		0.03(0.17)		0.02(0.08)	
CI-CPU		12.75	377.2		19.54		13.8		7.88		19.88		31.28		205.67	
#better		13	5		1		4		3		0					
#even		8	16		19		17		18		18					
#worse		0	0		1		0		0		3					

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(1 hour)	
		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)	0.55	**26**(26)	0.55
Mk03	204*	**204**(204)	**204**(204)	31	**204**(204)	24	**204**(204)	2	**204**	0	**204**(204)	0	**204**(204)	0.16	**204**(204)	0.16
Mk04	60*	**60**(60)	**60**(60)	13	**60**(60)	19	**60**(60)	10	**60**	0	**60**(60)	0	**60**(60)	0.47	**60**(60)	0.47
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**(58.15)	30.4	**57**(57.25)	268.54
Mk07	139*	**139**(139)	**139**(139)	26	**139**(141)	84	**139**(139)	33	**139**	20	**139**(140)	32	**139**(139.7)	61.58	**139**(139)	481.27
Mk08	523*	**523**(523)	**523**(523)	189	**523**(523)	83	**523**(523)	3	**523**	0	**523**(523)	0	**523**(523)	0.36	**523**(523)	0.36
Mk09	307*	**307**(307)	**307**(307)	123	**307**(307)	52	**307**(307)	24	**307**	1	**307**(307)	0	**307**(307)	1.13	**307**(307)	1.13
Mk10	189	197(198)	198(202)	266	196(197)	94	198(199)	104	197	33	197(199)	59	195(195.95)	36.78	**193**(194.6)	827.34
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(0.51)		0.23(0.34)	
CI-CPU		12.75	53.99		35.93		17.45		5.7		16.57		13.31		158.15	
#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	2.35	0.49	0.8	0.23	0.35	0.1	0.24
la26-la30	0.33	0.61	0.43	0.66	0.3	0.73	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		18.56		51.23		34.78		44.68		30.25		47.83		25.76	
#better	4		5				18		19				13		9			
#even	39		38				25		24				30		34			
#worse	0		0				0		0				0		0			

Table 5: The improved results of MAE compared with CPO on 45 instances

Ins.	CPO		MAE	Ins.	CPO		MAE
	LB	UB	UB		LB	UB	UB
Mk05	168	173	172	rdata-la23	816	832	831
Mk10	183	195	193	rdata-la24	775	805	795
02a	2228	2234	2228	rdata-la25	768	787	779
05a	2189	2213	2203	rdata-la26	1056	1066	1057
06a	2162	2191	2181	rdata-la27	1085	1099	1086
07a	2216	2277	2254	rdata-la28	1075	1079	1076
08a	2061	2066	2062	rdata-la29	993	1001	994
10a	2197	2263	2245	rdata-la30	1068	1089	1071
11a	2017	2067	2045	rdata-la31	1520	1522	1520
12a	1969	2013	2008	rdata-la32	1657	1658	1657
13a	2197	2258	2236	rdata-la33	1497	1498	1497
14a	2161	2163	2162	rdata-la34	1535	1536	1535
16a	2148	2240	2232	vdata-car1	5005	5006	5005
17a	2088	2140	2121	vdata-car3	5597	5599	5597
18a	2057	2125	2103	vdata-car5	4909	4912	4910
edata-abz7	564	620	610	vdata-la22	733	734	733
edata-abz8	586	639	636	vdata-la25	751	753	752
rdata-abz7	492	535	522	vdata-la29	993	994	993
rdata-abz8	506	558	535	vdata-la30	1068	1069	1068
rdata-abz9	497	553	536	vdata-la32	1657	1658	1657
rdata-car3	5597	5623	5622	vdata-la33	1497	1498	1497
rdata-la21	808	838	825	vdata-la35	1549	1550	1549
rdata-la22	741	755	753				

Comparison with the industrial solver Quintiq

We now compare MAE with the well-known industrial solver Quintiq, which has found new best-known solutions for 119 out of 313 instances. It also indicates the world records of solution quality for all the 313 instances, together with the first method to hit the records. However, Quintiq did not describe their methods and the time limits to obtain these results.

Table 6: New world records obtained by MAE

Ins.	Previous world record						MAE
	LB	LB Ref.	LB Date	UB	UB Ref.	UB Date	UB
05a	2192	[Q]	Jan. 2014	2204	[Q]	Nov. 2015	2203
07a	2216	[CPO]	Dec. 2014	2264	[Q]	Nov. 2015	2254
13a	2197	[CPO]	Dec. 2014	2239	[Q]	Jan. 2016	2236
rdata-abz7	493	[Q]	Mar. 2013	524	[Q]	Jan. 2016	522
rdata-abz8	507	[Q]	Mar. 2013	536	[Q]	Jan. 2016	535
rdata-la22	741	[CPO]	Dec. 2014	755	[CPO]	Nov. 2013	753
rdata-la23	816	[Q]	Feb. 2014	832	[CPO]	Mar. 2013	831
rdata-la24	775	[Q]	Feb. 2014	796	[Q]	Nov. 2015	795
rdata-la25	768	[CPO]	Dec. 2014	783	[Q]	Jan. 2016	779
vdata-car5	4909	[Q]	Mar. 2013	4911	[Q]	Nov. 2015	4910

We compare MAE with Quintiq by using a time limit of 1 hour for MAE. Experiments show that MAE obtains better results than Quintiq for 10 out of 121 instances. These new records obtained by MAE are provided in Table 6 for future comparison, where column "UB Ref." and "UB Date" represent the method and the date to obtain the new record, respectively, and "[Q]" represents the Quintiq method. Finally, the summarized comparison of MAE with CPO and Quintiq is reported in Table 7.

Discuss and analysis

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 $0.2 * |N_c|$

Table 7: Summary of MAE compared with CPO and Quintiq

Set		MAE(1 hour) vs CPO(8 hours)			MAE(1 hour) vs Quintiq		
		#better	#even	#worse	#better	#even	#worse
DPdata		13	5	0	3	2	10
BCdata		0	18	3	0	0	0
BRdata		2	8	0	0	2	0
HUdata	*edata*	2	60	4	0	20	0
	rdata	18	48	0	6	30	1
	vdata	10	53	3	1	22	0
	sdata	0	63	3	0	18	6
Total		45	255	13	10	94	17

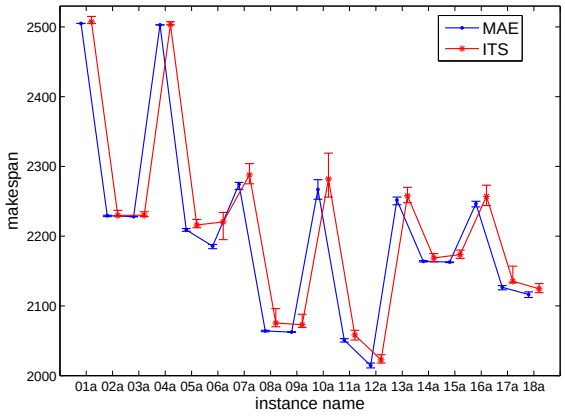

Figure 4: Comparison between MAE and ITS on *DPdata*.

moves in $N^{\pi} \cup N^{k}$ 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.

Fig. 4 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 the preliminary experiments, we have taken 11 different values (5, 6, ...,15) of p, 5 different groups of values ([0, 0.2], ..., [0.8, 1]) of $[\alpha, \beta]$, and 10 different values (1, ...,10) of γ to analyze the parameter sensitivity. There are totally 11 * 5 * 10 = 550 combinations for all the parameters. We ran MAE 10 times independently to solve a relatively difficult instance seti5xyz in *BCdata* with each of the 550 combinations, the cutoff time of each run being 90 seconds. The results show that MAE achieves the best performance when p, α, β, and γ are set to 10, 0.4, 0.6, and 5, respectively, considering both solution quality and computational efficiency. In the following paragraphs, we will further analyze the impact of one parameter on the performance of MAE

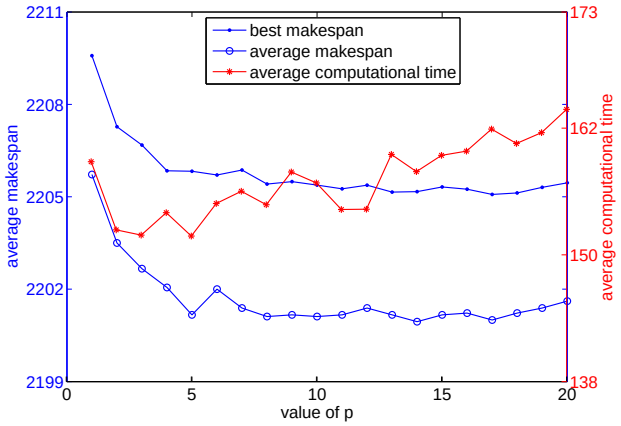

Figure 5: The average makespan and computational time corresponding to different values of parameter p on *DPdata*.

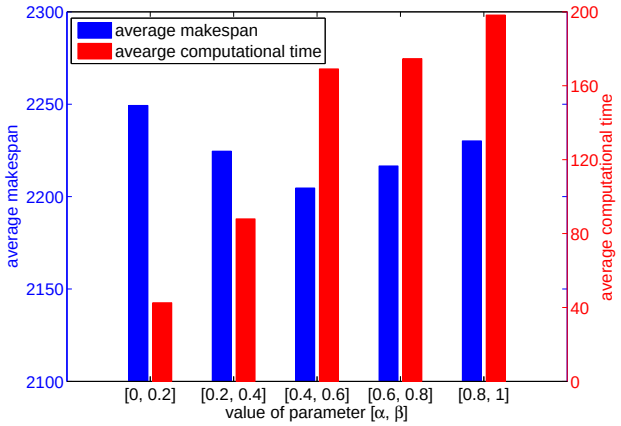

Figure 6: The average makespan and computational time corresponding to different values of parameter $[\alpha, \beta]$ on *DPdata*.

by extending its value domain and keeping other parameters fixed.

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. 5. One finds that the average makespan decreases when $p \in \{1, \ldots, 10\}$ and keeps flat or slightly increases when $p \in \{10, \ldots, 20\}$, while the computational time 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 in a cycle 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.

To analyze the impact of the parameters α, β, γ on MAE,

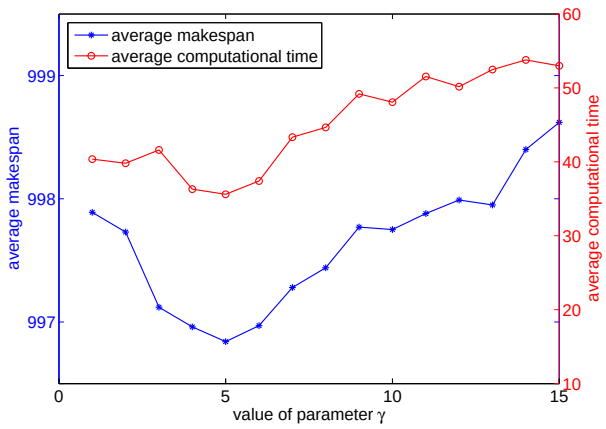

Figure 7: The average makespan and computational time corresponding to different values of parameter γ on *BCdata*.

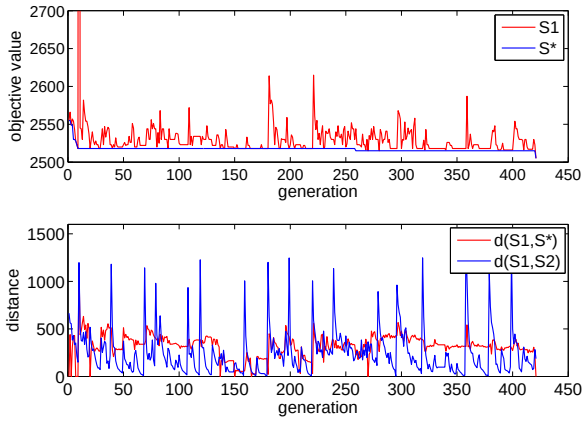

Figure 8: The evolution of objective values and distances when solving instance *01a*.

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 *DPdata* and *BCdata*, respectively. The results are presented in Fig. 6 and Fig. 7. Fig. 6 shows 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. Fig. 7 shows that both average makespan and computational time decrease when $\gamma \in \{1, \ldots, 5\}$ and increase when $\gamma \in \{5, \ldots, 15\}$. Considering both solution quality and computational efficiency, α, β, γ are suggested to be 0.4, 0.6, and 5, respectively.

MAE is effective because it maintains a good balance between intensification and diversification using two individuals S_1 and S_2, where S_1 plays the role of intensification and S_2 plays the role of diversification. To illustrate this, we depict in Fig. 8 the evolution of the objective value of S_1 and S^* which is the best S_1 obtained so far, and the distances $d(S_1, S^*)$ and $d(S_1, S_2)$ when solving the representative instance *01a*. We see that the objective value of S_1 is generally good and that of S^* is monotonously improved. This is possible because $d(S_1, S_2)$ periodically becomes large (comparing with $d(S_1, S^*)$), so that the path relinking operator on S_1 and S_2 is able to find diversified solutions.

Furthermore, we apply statistical significance test on the average makespan of the instances which are obtained by multiple runs of MAE compared with SSPR and GRASP-mELS, the resulting p-value of the average makespan between MAE and SSPR (GRASP-mELS) are reported in Table 8. Considering a level of significance of 0.05, one observes from Table 8 that there is significant difference between MAE and SSPR on *DPdata*, *rdata*, and *vdata*, and there is no significant difference between MAE and SSPR on *BCdata*, *BRdata*, and *edata*. Besides, there is significant difference between MAE and GRASP-mELS on *DPdata*, *rdata*, *vdata*, and *BCdata*, and there is no significant difference between MAE and GRASP-mELS on *BRdata* and *edata*. The reason may lie in the fact that the instances in *BCdata*, *BRdata*, and *edata* are relatively easy to solve. The values of #better, #even, and #worse in Tables 1-4 also give an idea of the difficulties of the instances.

Table 8: Statistical significance test

Benchmark set	MAE vs. SSPR	MAE vs. GRASP-mELS
DPdata	4.79×10^{-2}	3.28×10^{-5}
rdata	1.3×10^{-4}	6.17×10^{-5}
vdata	2.13×10^{-3}	3.69×10^{-4}
BCdata	8.37×10^{-2}	7.12×10^{-3}
BRdata	0.2066	0.2230
edata	0.7746	0.7724

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. We strongly believe that this two-individual based mater-apprentice evolutionary algorithm is a promising framework for solving other challenging combinatorial optimization problems.

Acknowledgments

The research was supported by China Postdoctoral Science Foundation funded project under grant number 2018M630861 and National Natural Science Foundation of China under grant numbers 61370183 and71320107001 .

References

Barnes, J. W., and Chambers, J. B. 1998. Flexible job shop scheduling by tabu search. Technical report, The University of Texas at Austin.

Bożejko, W.; Uchroński, M.; and Wodecki, M. 2010. Parallel hybrid metaheuristics for the flexible job shop problem. *Computers and Industrial Engineering* 59(2):323–333.

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

Brucker, P., and Schlie, R. 1991. *Job-shop scheduling with multi-purpose machines*. Springer-Verlag New York, Inc.

Dauzère-Pérès, S., and Paulli, J. 1997. 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.

Duarte, A.; Sánchez, A.; Fernández, F.; and Cabido, R. 2005. A low-level hybridization between memetic algorithm and vns for the max-cut problem. In *Proceedings of the 7th Annual Conference on Genetic and Evolutionary Computation*, GECCO '05, 999–1006. New York, NY, USA: ACM.

Gao, K.-Z.; Suganthan, P. N.; Pan, Q.-K.; Chua, T. J.; Cai, T.-X.; and Chong, C.-S. 2016. Discrete harmony search algorithm for flexible job shop scheduling problem with multiple objectives. *Journal of Intelligent Manufacturing* 27(2):363–374.

Gao, J.; Sun, L.; and Gen, M. 2008. A hybrid genetic and variable neighborhood descent algorithm for flexible job shop scheduling problems. *Computers & Operations Research* 35(9):2892–2907.

Garey, M. R.; Johnson, D. S.; and Sethi, R. 1976. The complexity of flowshop and jobshop scheduling. *Mathematics of Operations Research* 1(2):117–129.

Gomes, M. C.; Barbosa-Pvoa, A. P.; and Novais, A. Q. 2013. 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.

González, M. A.; Vela, C. R.; and Varela, R. 2013. An efficient memetic algorithm for the flexible job shop with setup times. In *International Conference on International Conference on Automated Planning and Scheduling*, 91–99.

González, M. A.; Vela, C. R.; and Varela, R. 2015. Scatter search with path relinking for the flexible job shop scheduling problem. *European Journal of Operational Research* 245(1):35–45.

Gutiérrez, C., and García-Magariño, I. 2011. Modular design of a hybrid genetic algorithm for a flexible jobshop scheduling problem. *Knowledge-Based Systems* 24(1):102–112.

Hansmann, R. S.; Rieger, T.; and Zimmermann, U. T. 2014. Flexible job shop scheduling with blockages. *Mathematical Methods of Operations Research* 79(2):135–161.

Hmida, A. B.; Haouari, M.; and Lopez, P. 2010. Discrepancy search for the flexible job shop scheduling problem. *Computers & Operations Research* 37(12):2192–2201.

Hurink, J.; Jurisch, B.; and Thole, M. 1994. Tabu search for the job-shop scheduling problem with multi-purpose machines. *Operations-Research-Spektrum* 15(4):205–215.

Kemmoé-Tchomté, S.; Lamy, D.; and Tchernev, N. 2017. An effective multi-start multi-level evolutionary local search for the flexible job-shop problem. *Engineering Applications of Artificial Intelligence* 62:80–95.

Lahiri, M., and Cebrian, M. 2010. The genetic algorithm as a general diffusion model for social networks. In *Twenty-Fourth AAAI Conference on Artificial Intelligence*, 494–499.

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

Li, J.-Q.; Pan, Q.-K.; and Liang, Y.-C. 2010. An effective hybrid tabu search algorithm for multi-objective flexible job-shop scheduling problems. *Computers & Industrial Engineering* 59(4):647–662.

Lü, Z.; Glover, F.; and Hao, J. K. 2010. A hybrid metaheuristic approach to solving the UBQP problem. *European Journal of Operational Research* 207(3):1254–1262.

Mastrolilli, M., and Gambardella, L. M. 2000. Effective neighborhood functions for the flexible job shop problem. *Journal of Scheduling* 3(1):3–20.

Moalic, L., and Gondran, A. 2017. Variations on memetic algorithms for graph coloring problems. *Journal of Heuristics* 1–24.

Oddi, A.; Rasconi, R.; Cesta, A.; and Smith, S. F. 2011. Iterative flattening search for the flexible job shop scheduling problem. In *International Joint Conference on Artificial Intelligence*, 1991–1996.

Özgüven, C.; Özbakr, L.; and Yavuz, Y. 2010. Mathematical models for job-shop scheduling problems with routing and process plan flexibility. *Applied Mathematical Modelling* 34(6):1539–1548.

Palacios, J. J.; González, M. A.; Vela, C. R.; González-Rodríguez, I.; and Puente, J. 2015. Genetic tabu search for the fuzzy flexible job shop problem. *Computers & Operations Research* 54(C):74–89.

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

Pezzella, F.; Morganti, G.; and Ciaschetti, G. 2008. A genetic algorithm for the flexible job-shop scheduling problem. *Computers & Operations Research* 35(10):3202–3212.

Roshanaei, V.; Azab, A.; and Elmaraghy, H. 2013. Mathematical modelling and a meta-heuristic for flexible job shop scheduling. *International Journal of Production Research* 51(20):6247–6274.

Sutton, A. M., and Neumann, F. 2012. A parameterized runtime analysis of evolutionary algorithms for the euclidean traveling salesperson problem. In *AAAI Conference on Artificial Intelligence*, 595 – 628.

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

Vilím, P.; Laborie, P.; and Shaw, P. 2015. Failure-directed search for constraint-based scheduling. In *International Conference on AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, 437–453.

Wang, L.; Zhou, G.; Xu, Y.; Wang, S.; and Liu, M. 2012. An effective artificial bee colony algorithm for the flexible job-shop scheduling problem. *International Journal of Advanced Manufacturing Technology* 56(60):1–8.

Wang, L.; Zhou, G.; Xu, Y.; and Liu, M. 2013. A hybrid artificial bee colony algorithm for the fuzzy flexible job-shop scheduling problem. *International Journal of Production Research* 51(12):3593–3608.

Yu, Y.; Yao, X.; and Zhou, Z.-H. 2013. 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*, 3190–3194. AAAI Press.

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

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