Flexible Job Shop Scheduling using a Multiobjective Memetic Algorithm

Tsung-Che Chiang and Hsiao-Jou Lin,

Department of Computer Science and Information Engineering,
National Taiwan Normal University, Taiwan, R.O.C.
tcchiang@ieee.org, sirsene@gmail.com

The official publication is available at www.springerlink.com

Abstract. This paper addresses the flexible job shop scheduling problem with minimization of the makespan, maximum machine workload, and total machine workload as the objectives. A multiobjective memetic algorithm is proposed. It belongs to the integrated approach, which deals with the routing and sequencing sub-problems together. Dominance-based and aggregation-based fitness assignment methods are used in the parts of genetic algorithm and local search, respectively. The local search procedure follows the framework of variable neighborhood descent algorithm. The proposed algorithm is compared with three benchmark algorithms using fifteen classic problem instances. Its performance is better in terms of the number and quality of the obtained solutions.

Keywords: multiobjective, Pareto optimal, flexible job shop scheduling, memetic algorithm, variable neighborhood descent

1 Introduction

The flexible job shop is an extension of the classical job shop. There are m machines in the shop, and n jobs are to be processed. Each job j consists of n_j operations. Each operation o_{ji} of job j should be processed by one machine in the set of eligible machines M_{ji}. The existence of multiple eligible machines brings flexibility to the job shop and thus gets the name of flexible job shop. If each machine can process all operations, the shop is referred to as totally flexible; otherwise, it is partially flexible. The processing time of operation o_{ji} on a machine $k \in M_{ji}$ is a constant p_{jik} and is known a priori. Each machine can process only one job at a time, and the processing is un-interruptible. Processing of operations should follow the precedence constraint, namely, the processing of operation $o_{j(i+1)}$ cannot start until the processing of operation o_{ji} is finished. In this paper, we address the scheduling problem in a static flexible job shop, in which all jobs are ready at time zero.

Scheduling in the flexible job shop is more complicated than in the classical job shop, which is already known to be NP-hard. Classical job shop scheduling requires sequencing of operations on machines, whereas flexible job shop scheduling needs to deal with not only sequencing but also assigning operations to suitable machines (routing).

The most common objective in production scheduling is to minimize the makespan, which is the time required to complete all jobs. Proper sequencing of operations can shorten the makespan and thus provides quick response to the market demand and high machine utilization. In the flexible job shop, objectives like reducing and balancing the machine workload are also common. Proper assignment of machines to operations can utilize machines efficiently and protect machines from overuse. Different routing decisions also turn the flexible job shop scheduling problem (FJSP) into different job shop scheduling problems (JSP) and have the impact on the makespan. In this study, we consider three objective functions: makespan (C_M), total workload (sum of workload of all machines, W_T), and maximum workload (over all machines, W_M). All objective functions are to be minimized.

Common approaches to multiobjective optimization problems can be roughly divided into two groups: a priori and Pareto approaches. A priori approaches usually define an aggregation function (e.g. linear weighted sum) to convert multiple objective functions into a single objective function or define a preference order to optimize the objective functions one by one. It is applicable when users can define the aggregation function or preference order, and its goal is to find a single optimal solution. On the other hand, when users feel difficult to define the relationship between objective functions a priori, Pareto approaches are useful. These approaches evaluate solution quality based on the concept of Pareto dominance. A solution x is said to dominate a solution y if and only if x is not worse than y in all objective functions and is better than y in at least one objective function. The solutions that are not dominated by any

other solution are called Pareto optimal. The goal of Pareto approaches is to find or approximate the set of Pareto optimal solutions. Although users still have to select a favorite solution from the Pareto optimal solutions, the distribution of these solutions on the objective space is helpful for users to make a suitable decision. Besides, the number of the Pareto optimal solutions is usually much smaller than the number of all solutions, and making a decision becomes easier.

The rest of this paper is organized as follows. Section 2 gives a literature review on FJSP. Section 3 details the proposed multiobjective memetic algorithm (MA). Experiments and results are presented in section 4, and conclusions and future work are given in Section 5.

2 Literature Review

As mentioned in last section, the FJSP consists of two sub-problems: routing and sequencing. Brandimarte [1] generated an initial solution by dispatching rules and fixed the routing decisions. A tabu search (TS) was then developed to improve the initial solution by solving the derived JSP. The neighborhood function was to exchange adjacent jobs on the critical path. An advanced method was developed by using a high-level TS to do re-routing: inserting all operations on the critical path to all possible positions on all eligible machines. Among these solutions the best one (the solution with the minimal makespan) was selected to start the low-level TS to solve the sequencing sub-problem again. In this approach, routing and sequencing sub-problems are solved separately and alternatively. It is a representative of the so-called hierarchical approach. Hurink *et al*. [2] addressed a special case of FJSP, where the processing times on all eligible machines are identical. Their approach is also based on TS but is different from Brandimarte's in several points: (1) the initial solution was generated by a sophisticated procedure based on beam search; (2) the neighborhood size was reduced; (3) the neighborhood consisted of solutions with changes of both routing and sequencing decisions. By solving routing and sequencing sub-problems concurrently, this kind of approach is referred to as the integrated approach. The neighborhood function is important to search-based algorithms like TS. Bad neighborhood functions waste computational effort on useless solutions and even do not lead to the optimal one. Mastrolilli and Gambardella [3] proposed two effective neighborhood functions by identifying two special groups of operations L and R and inserting critical operations between operations in L\R and R\L. Their TS was able to find good solutions with very limited number of iterations. Bozejko *et al*. [4] also focused on the reduction of neighborhood. Their approach was a hierarchical approach and was parallelized on the graphics processing units (GPU) environment.

Ho and Tay [5] minimized the makespan using a cultural algorithm. Two belief spaces, the operation belief space and chromosome belief space, were built for effective mutation and environmental selection. Half of the initial population was generated by applying the dispatching rules, and the remaining half was generated randomly. Later, Tay and Ho [6] relied on genetic programming (GP) to produce better dispatching rules. The routing decision was made by assigning operations to the machine with the shortest waiting time, and the sequencing decision was made by selecting the highest-priority job based on the GP-generated dispatching rule. Pezzella *et al*. [7] tackled the FJSP by a genetic algorithm (GA). A large amount of domain knowledge was used to generate the initial population. The routing sub-problem was solved by the approach by localization (AL) [8], and the sequencing sub-problem was solved by three dispatching rules. One crossover and one mutator were proposed for sequencing, and one crossover and two mutators were for routing. Yazdani *et al*. [9] proposed a parallel variable neighborhood search (VNS) algorithm. They applied the approach by localization and random sequencing to generate a large number of solutions and then selected the best one as the initial solution. They used nine neighborhood functions and a master-slave parallelization model. Bagheri *et al*.' artificial immune algorithm [10] mainly followed Pezzella *et al*.' approach and introduced a reordering mutation and random immigration.

Kacem *et al*. [8] addressed the FJSP by considering the makespan, total workload, and maximum workload. They proposed the AL, which is widely used by in later research studies as a promising method to do the machine assignment. They also proposed a GA to improve the machine assignment and did sequencing of operations by dispatching rules. Their problem instances initiated the research on multiobjective FJSP (MOFJSP). Xia and Wu [11] solved the MOFJSP by aggregating the three objective values using the linear weighted sum function. Their approach hybridized particle swarm optimization (PSO) and simulated annealing (SA). The PSO dealt with the routing sub-problem, and the SA dealt with the sequencing sub-problem. Better solutions were found for two of Kacem *et al*.' instances. Gao *et al*. [12] proposed a hybrid GA to solve the MOFJSP. They also aggregated the three objective values through linear weighted summation. The local search procedure used two kinds of neighborhood functions, one swapping specific critical operations [13] and one moving critical operations to alternative machines. Their approach showed better performance than approaches in [3], [8], and [11]. Later, Gao *et al*. [14] focused on only insertion-based neighborhood function in

the local search procedure and reduced the neighborhood size by eliminating the useless moves. To insert a critical operation, only the positions that do not violate the earliest completion time of the preceding operations and the latest start time of the succeeding operations are considered. A comprehensive experiment was conducted on 181 problem instances, and the proposed algorithm produced very good results. Zhang *et al*. [15] developed a hybrid of PSO and TS to solve the MOFJSP. The objective values were again aggregated by linear weighted summation. The TS used the neighborhood proposed in [3]. Xing *et al*. [16] tackled the MOFJSP with the aggregated objective based on the ant colony optimization (ACO) algorithm, local search, and dispatching rules. The ACO made the routing and sequencing decisions, and a local search was conducted whenever the best solution was updated. The local search moves each operation to the machines with the minimal or second-minimal processing time or minimal total processing time and then sequences the operations by five dispatching rules. Li *et al*. [17] used a similar procedure to Pezzella *et al*.' one [7] to generate the initial solution. Then, a TS and a hill climbing were conducted to do machine assignment and operation sequencing, respectively. The neighborhood in the TS considered reducing the number of critical operations, and the neighborhood in the hill climbing focused on the swap and insertion of public critical operations.

Although Kacem *et al*.' study [8] has attracted many researchers' attention to the MOFJSP [11]-[17], most studies used the linear weighted summation to aggregate the concerned objectives and aimed at finding a single optimal solution with respect to a fixed set of weight values. For example, Gao *et al*. [12] set the weights by (0.85, 0.1, 0.05) and Xing *et al*. [16] set by (0.5, 0.3, 0.2) for the makespan, maximum workload, and minimal workload, respectively. Until very recently, some researchers started to solve the MOFJSP in a Pareto way, namely, with the goal of finding the set of Pareto optimal solutions. Xing *et al*. [18] utilized a local search to optimize the machine assignment and applied dispatching rules for operation sequencing. They still aggregated the objective values by linear weighted summation, but ten different sets of weights were used to seek for the set of Pareto optimal solutions. The experimental results showed that multiple Pareto optimal solutions do exist for the MOFJSP. Moslehi and Mahnam [19] proposed a PSO hybridizing a local search. An archive was used to store the non-dominated solutions found during the search process. In order to find the set of Pareto optimal solutions, they used the sigma method [20] to select a local guiding particle from the archive for each particle in the current swarm. The parameterized active schedule decoder in [21] was also used. Their local search focused on moving only critical operations.

The FJSP is a problem of high complexity and practical value, and it has been widely investigated in last two decades. Researches on its multiobjective version have also grown rapidly. However, solving the MOFJSP in the Pareto point of view is still at the infancy. We would like to know if the concerned objectives are really conflicting with each other. We also want to compare the performance of approaches based on aggregation functions and Pareto dominance. Moreover, benchmark solutions for the MOFJSP instances are eagerly required to encourage more research studies and facilitate performance validation. All of the above motivate this study.

3 Proposed Multiobjective Memetic Algorithm

The proposed MA is outlined as follows. Details of each step will be given in the following subsections.

1. *Initialization*: Generate the initial population of size N_{POP}. Decode the initial solutions, calculate the objective values, and assign the fitness values. Set the generation number $t = 1$.
2. *Reproduction*: Generate the population at generation t+1 through mating selection, crossover, and mutation. N_{POP} offspring are generated, and the best N_{POP} individuals among the ($2 \cdot N_{POP}$) individuals will survive.
3. *Duplicate elimination*: If there are more than two individuals having the same objective values, replace the duplicates by applying mutation five times.
4. *Local search*: If t is a multiple of T_{LS} (interval to do local search), do local search to the best N_{LS} individuals. Otherwise, go to Step 5.
5. *Termination*: If $t > T_{GEN}$, stop; otherwise, $t = t + 1$, go to Step 2.

3.1 Chromosome Encoding and Decoding

Each chromosome is a permutation of 3-tuples (j, i, k), in which j, i, and k represent the indices of job, operation, and machine, respectively. A tuple (j, i, k) means that the operation o_{ji} will be processed in machine k. The position of each tuple determines the priority of the corresponding operation. The tuple positioned closer to the left end has a higher priority. Taking Fig. 1 as an example, it encodes the sequence of six operations, three of job 1 and three of job 2. The

first operation of job 2 and the first operation of job 1, o_{21} and o_{11}, are processed by machine 2. The operation o_{21} has higher priority than o_{11}.

(2, 1, 2)	(1, 1, 2)	(2, 2, 1)	(2, 3, 3)	(1, 2, 3)	(1, 3, 2)

Fig. 1. Chromosome encoding.

To decode a chromosome into a feasible schedule, firstly, the machines are assigned to operations according to the information encoded on the chromosome. Then, the Giffler and Thompson (GT) algorithm [23] is used to build the schedule. The GT algorithm repeats the following steps: (1) Identify the operation with the earliest completion time *ect* and its destination machine m^*; (2) Collect all operations assigned to m^* and startable no later than *ect* to form the conflicting set of operations O; (3) Schedule the operation in O with the highest priority; (4) Update the earliest completion times of affected operations (the succeeding operation of the scheduled one and the operations assigned to m^*). The permutation in the chromosome is re-ordered according to the order in which the operations are scheduled during the GT algorithm. After a schedule is obtained, the concerned objective values are calculated.

3.2 Fitness Assignment and Selection

There are two types of selection in the evolutionary algorithm, the mating selection and environmental selection. The former determines which individuals will do crossover to produce the offspring, whereas the latter decides which individuals will survive to the next generation. Selection is usually done based on the fitness value. There are many good fitness assignment methods for multiobjective optimization, for example, NSGA-II [22] and SPEA2 [24]. In this study we assign the fitness based on NSGA-II. Given the rank r and crowding distance d, the fitness is assigned by $1/(r+1/(d+2))$. (The non-dominated solution has rank 1.) We use 2-tournament for mating selection, which selects two random individuals and take the one with higher fitness as a parent. Environmental selection follows the (μ, λ) mechanism, where $\mu = \lambda$. We do mating selection $N_{POP}/2$ times to select $N_{POP}/2$ pairs of parents. Each pair of parents produce two offspring through crossover. Offspring get mutated in probability p_m. Among the N_{POP} individuals in the current population and the N_{POP} offspring, the N_{POP} individuals with higher fitness will survive to the next generation.

3.3 Crossover and Mutation

Since the chromosome contains information on both routing and sequencing decisions, the crossover and mutation operators should be able to exchange and adjust both kinds of information. The assignment crossover (ASX) exchanges the machines of four randomly selected operations. The Precedence Preserving Order-based Crossover (POX) [25] selects one job randomly and keeps positions of its operations unchanged in each of the parents. All the remaining operations on each parent are re-positioned in the order they appear in the other parent. When two parents do crossover, one of ASX and POX is randomly chosen to generate the offspring. Two heuristic mutations are used to adjust the machine assignment. (In fact, the operation sequence is also affected.) They identify the machine m^{max} with the maximum makespan (workload) and then select an operation o_{ji} on m^{max} randomly. The operation o_{ji} is assigned to the machine m^{min} with the minimum makespan (workload). If m^{min} is not eligible for processing o_{ji}, o_{ji} is assigned to a random machine in the eligible machines M_{ji}.

3.4 Local Search

Although GAs have been applied to solve many optimization problems successfully, they are also known for lack of the ability of exploitation. The MA combines the GA and local search to gather their advantages to improve the search ability. In the proposed MA, we do local search to the best N_{LS} individuals in the population of GA every T_{LS} generations. The local search procedure is based on the framework of variable neighborhood descent (VND) algorithm [26]. We use three neighborhood functions, all focused on critical operations. Critical operations are the operations on the critical path, which is the longest path on the disjunctive graph representation of a schedule. If more than one critical path exists, we select one randomly. Table 1 gives the pseudo code.

Table 1. Pseudo code of the local search procedure

```
LocalSearch(Individual x)
Begin
    y* = x
    For t = 1 to T_SH
        If t = 1 Then y = x
        Else y = Shaking(x)

        For n = 1 to 3 // three neighborhood functions
            y' = SteepestDescent(y, N_n, n_E)
            If WS(y') ≤ WS(y) Then y = y'
            If WS(y') ≤ WS(y*) Then y* = y'
        End
    End
    x = y*
End
```

The first neighborhood (N_1) follows the neighborhood proposed in [13]. It swaps the first two and last two operations in the critical blocks. (A critical block is a sequence of consecutively processed critical operations on the same machine.) For the first (last) critical block, only the last (first) two operations are swapped. The second neighborhood (N_2) selects one random critical operation and one random eligible machine. Then, the operation is inserted into all possible positions on the machine. The third neighborhood (N_3) is an extension of N_2. It inserts the selected operation into all possible positions on all eligible machines. We apply the three neighborhood functions one by one using the steepest descent algorithm until the local optimum is reached. The local search aims at minimizing the makespan, and thus the fitness function WS(.) here is a linear weighted summation: $WS(y) = 0.8 \cdot C_M(y) + 0.1 \cdot W_M(y) + 0.1 \cdot W_T(y)$. We allow accepting equal-fitness neighboring solution for at most n_E times consecutively. Shaking of a solution is achieved by inserting a random critical operation to a random position on a random eligible machine.

3.5 Initial Population and Stopping Criterion

The initial population is generated by the procedure used in [7]. Routing decisions are made by two variants of the AL approach [8]. The global assignment variant (GAL) assigns operations to the machine with the minimal accumulated workload considering all operations simultaneously, whereas the local assignment variant (LAL) considers only one random operation at a time. Sequencing decisions are made by one of three dispatching rules, Random, Most Work Remaining (MWR), and Most number of Operations Remaining (MOR). Since GAL generates relatively deterministic results, routing decisions of 90% initial individuals are made by LAL to increase population diversity. Each dispatching rule is used to initialize 1/3 population. The proposed MA stops when a maximum number of generations (T_{GEN}) is reached.

4 Experiments and Results

4.1 Benchmark Problem Instances and Algorithms

We tested the proposed algorithm on two sets of problem instances from Kacem *et al*. [27] and Brandimarte [1]. Kacem *et al*.' data set contains five instances, whose scale ($n \times m$, n: number of jobs, m: number of machines) ranges from 4×5 to 15×10. Brandimarte's data set contains 15 instances, but only the first 10 instances are considered here, as most existing studies also did experiments on these instances only. The problem scale ranges from 10×6 to 20×15. These two data sets are the most commonly adopted benchmark instances in the literature on FJSP.

We compared our algorithm with three recent studies, Xing *et al*. in 2009 [18], Li *et al*. in 2010 [17], and Moslehi and Mahnam in 2011 [19].

4.2 Parameter Setting and Problem Analysis

We tested four combinations of population size (N_{POP}) and the interval of doing local search (T_{LS}). The parameter settings of these four variants (N_{POP}, T_{LS}) are (50, 500), (200, 500), (400, 25), and (400, 1), The maximum number of generations (T_{GEN}) was 500, and thus setting T_{LS} by 500 means that no local search is conducted. The first two variants are pure GA, and the last two variants conduct local search for 20 and 500 times, respectively. These four variants require different amount of computation effort. Different values of T_{LS} represent different levels of intensification. Values of other parameters, the mutation rate (p_m, see section 3.2), the number of individuals to do local search (N_{LS}, see section 3.4), times of shaking in local search (T_{SH}, see section 3.4) and the maximum number of consecutive equal-fitness moves in local search (n_E, see section 3.4), were set by 0.5, 10, 2, and 10, respectively based on some pilot runs considering solution quality and computation time.

We ran each variant to solve each problem instance for ten times. For each instance i, the non-dominated solutions among solutions obtained by all four variants over ten runs are collected as the reference set PF_i^*. Performance of each variant is measured by the number of non-dominated solutions in PF_i^*. Denote the solutions obtained by variant j at run k for problem instance i by PF_{ijk}. We calculated the average number of non-dominated solutions found by each run $N_{avg}(i, j)$ and the number of non-dominated solutions found by ten runs $N_{best}(i, j)$. These two measures stand for the average-case and best-case performance. Table 2 summarizes the results. Some observations are made in the following.

$$N_{avg}(i,j) = \frac{1}{10}\sum_{k=1}^{10}\left|PF_{ijk} \cap PF_i^*\right| \tag{1}$$

$$N_{best}(i,j) = \left|\left(\bigcup_{k=1\ldots10} PF_{ijk}\right) \cap PF_i^*\right| \tag{2}$$

Table 2. Performance of four variants (N_{POP}, T_{LS}) of the proposed MA.

Problem instance (n×m)	#non-dominated solution ($\|PF_i^*\|$)	Variant 1 (50, 500)	Variant 2 (200, 500)	Variant 3 (400, 25)	Variant 4 (400, 1)
Kacem 4×5	4	4.0[1] / 4[2]	4.0 / 4	4.0 / 4	3.8 / 4
Kacem 8×8	4	4.0 / 4	4.0 / 4	4.0 / 4	4.0 / 4
Kacem 10×7	3	2.8 / 3	3.0 /3	3.0 / 3	3.0 / 3
Kacem 10×10	4	1.8 / 4	3.1 / 4	3.5 / 4	3.5 /4
Kacem 15×10	2	0.0 / 0	0.3 / 2	0.8 / 2	1.6 / 2
Mk01 (10×6)	10	3.5 / 7	5.1 / 7	6.3 / 10	6.7 / 8
Mk02 (10×6)	8	3.4 / 5	4.1 / 6	4.7 / 6	4.7 / 8
Mk03 (15×8)	17	14.5 / 17	16.5 / 17	15.3 / 16	3.6 / 8
Mk04 (15×8)	23	13.6 / 16	15.6 / 16	16.5 / 20	9.8/ 17
Mk05 (15×4)	11	8.9 / 10	10 / 10	10.4 / 11	8 / 10
Mk06 (10×15)	127	0 / 0	2.6 / 23	8 / 50	10.8 / 91
Mk07 (20×5)	16	10.9 / 15	14.5 / 15	12.2 / 15	8.4 / 11
Mk08 (20×10)	9	9.0 / 9	9.0 / 9	9.0 / 9	8.3 / 9
Mk09 (20×10)	57	0.0 / 0	8.6 / 13	15.9 / 40	11.8 / 37
Mk10 (20×15)	278	0.0 / 0	0.1 / 1	9.9 / 95	19 / 187

[1] N_{avg}, average number of non-dominated solutions found by each run
[2] N_{best}, number of non-dominated solutions found by ten runs

First, the number of non-dominated solutions is varying between the tested problem instances. For some instances like Kacem 15×10, few non-dominated solutions are found, which implies that there is little conflict between the concerned objectives. For other instances like Mk06, the objectives are very conflicting and consequently a lot of non-dominated solutions are found. Even though the problem scale is similar (Mk08 and Mk09 have the same numbers of jobs and machines, and the numbers of operations are close.), the number of non-dominated solutions can be very different. More investigation on the relationship between the number of non-dominated solutions and the problem characteristics including degree of flexibility (number of eligible machines) and variation of processing times among eligible machines is an interesting topic for future work.

Second, the problem difficulty and the required computational effort are also varying. For each problem instance, the variant with the best average-case performance is marked by gray color. Ties are broken by the best-case performance and the computational effort. Instances like Kacem 4×5 and Mk08 can be easily solved by the simplest variant, but instances like Mk06 and Mk10 need the variant with large population size and frequent local search. Another interesting observation is that variants with more computational effort do not necessarily perform better than those with less computational effort. For example, variant 4 is much worse than variant 2 on instance Mk03. Too large population may slow down the search process, and too frequent local search may cause premature convergence. Fitness landscape analysis on these instances will also be a good research direction to follow.

Third, benchmark problem instances are important for evaluating algorithms. We would like to develop an algorithm suitable for a wide class of problem instances instead of instances with very specific features. With the variety in terms of conflict between objectives (number of non-dominated solutions) and difficulty (requirement of population size and times of local search), this data set is suitable for testing algorithms for MOFJSP.

4.3 Performance Comparison

We compare the proposed MA with three benchmark algorithms [17]-[19]. Since the non-dominated solutions found by these algorithms are already provided in the papers, we do not re-implement these algorithms but compare the results directly. All three papers list the non-dominated solutions for five Kacem *et al*' instances, except that Moslehi and Mahnam [19] did not solve the 4×5 and 10×7 instances. We count the number of non-dominated solutions found by the four algorithms and show the results in Table 3.

The results indicate that the proposed MA finds all non-dominated solutions for all Kacem *et al*.' instances. None of the three benchmark algorithms can achieve this. Xing *et al*. [18] used the linear weighted summation to aggregate three objectives and ran their algorithm ten times with different weight sets to collect the non-dominated solutions. However, the multiple trials cannot guarantee the discovery of all non-dominated solutions. It shows the difficulty of setting proper weights on the objectives. Besides, the aggregation-based method needs multiple runs and more computation time. Although Moslehi and Mahnam [19] maintained an archive of non-dominated solutions and obtained five and two solutions for 8×8 and 15×10, some of their solutions are dominated by ours.

Table 3. Comparison of the proposed MA with three recent studies on five instances in [27]

Problem instance	#non-dominated solution ($\|PF_i^*\|$)	Xing2009 [18]	Li2010 [17]	Moslehi2011 [19]	Proposed
4×5	4	3	2	n/a	4
8×8	4	3	2	3	4
10×7	3	3	2	n/a	3
10×10	4	3	3	4	4
15×10	2	1	2	1	2

Performance of the proposed MA on Mk01-Mk10 instances is compared only with Li *et al*. [17] since their results are better than Xing *et al*. [18] and Moslehi and Mahnam [19] did not solve these instances. Li *et al*. listed only one solution for each instances, and thus we also selected one solution from the set of non-dominated solutions found by our MA. From Table 4, our MA finds solutions dominating the solutions in [17] for four instances, but our MA spends more computation time. Since the computation time depends on the implementation of the programs, we provide the number of evaluations as a reference. More experiments on testing the algorithms with different time limits as the stopping criteria are left as our future work.

Table 4. Comparison of the proposed MA with Li *et al.*' study [17] on 10 instances in [1]

	Li 2010				Proposed				
	C_M	W_M	W_T	Time (s)	C_M	W_M	W_T	Time (s)	#Eval ($\times 10^3$)
Mk01(10×6)	40	36	167	2	40	36	167	4	43
Mk02(10×6)	26	151	26	16	26	151	26	48	416
Mk03(15×8)	204	852	204	8	**204**	**850**	**204**	7	44
Mk04(15×8)	61	366	61	43	61	373	60	61	429
Mk05(15×4)	172	687	172	15	172	687	172	84	476
Mk06(10×15)	65	398	62	64	**65**	**392**	**61**	31	151
Mk07(20×5)	140	695	140	36	140	695	140	30	171
Mk08(20×10)	523	2524	523	4	523	2524	523	15	61
Mk09(20×10)	310	2294	301	66	**307**	**2273**	**306**	120	410
Mk10(20×15)	214	2053	210	121	**208**	**2032**	**205**	658	2824

4.4 List of Non-dominated Solutions

We provide the list of non-dominated solutions found by our MA in Table 5. The list can serve as a benchmark for researchers who are interested in solving MOFJSP. From the results, we make a further observation. The makespan and maximum workload are positively correlated for most instances. For several instances, the makespan and maximum workload are even equal. It implies that minimizing the maximum workload is helpful to minimize the makespan. Since the number of non-dominated solutions is too large for some instances, we have to omit some of the solutions to save the space. A complete list is available on the first author's website[1].

5 Conclusions and Future Work

The flexible job shop is an extension of the classical job shop. Scheduling in the flexible job shop consists of the routing and sequencing sub-problems. The introduction of the routing sub-problem also raises the objectives of minimizing the maximum workload and total workload. Most studies deal with the MOFJSP by the aggregation-based methods and seek for only a single optimal solution with respect to a certain function of objective values. In this study we proposed a multiobjective MA to solve the MOFJSP in a Pareto way. We aim at generating the set of Pareto optimal solutions so that decision makers do not need to define the aggregation function in advance and can select the favorite solution based on the trade-off between objective values. Comparing with three algorithms on 15 instances, our MA provides better results in terms of number and quality of solutions. Future work includes the investigation of problem characteristics, more comprehensive experiments on performance comparison, and neighborhood functions for minimizing the maximum workload.

References

1. Brandimarte, P.: Routing and Scheduling in a Flexible Job Shop by Tabu Search. Annals of Operations Research 41, 157 – 183 (1993)
2. Hurink, J., Jurisch, B., and Thole, M.: Tabu Search for the Job-Shop Scheduling Problem with Multi-purpose Machines. OR Spektrum 15, 205 – 215 (1994)
3. Mastrolilli, M., Gambardella, L.M.: Effective Neighborhood Functions for the Flexible Job Shop Problem. Journal of Scheduling 3, 3 – 20 (2000)
4. Bozejko, W., Uchronski, M., Wodecki, M.: Parallel Hybrid Metaheuristics for the Flexible Job Shop Problem. Computers & Industrial Engineering 59, 323 – 333 (2010)
5. Ho, N.B., Tay, J.C.: GENACE: An Efficient Cultural Algorithm for Solving the Flexible Job-shop Problem. In: Proceedings of the IEEE Congress on Evolutionary Computation, pp. 1759 – 1766 (2004)
6. Tay, J.C., Ho, N.B.: Evolving Dispatching Rules using Genetic Programming for Solving Multi-objective Flexible Job-shop Problems. Computers & Industrial Engineering 54, 453 – 473 (2008)

[1] http://web.ntnu.edu.tw/~tcchiang

7. Pezzella, F., Morganti, G., Ciaschetti, G.: A Genetic Algorithm for the Flexible Job-shop Scheduling Problem. Computers & Operations Research 35, 3202 – 3212 (2008)
8. Kacem, I., Hammadi, S., Borne, P.: Approach by Localization and Multiobjective Evolutionary Optimization for Flexible Job-shop Scheduling Problem. IEEE Transactions on Systems, Man, and Cybernetics – Part C 32, 1 – 13 (2002)
9. Yazdani, M., Amiri, M., Zandieh, M.: Flexible Job-shop Scheduling with Parallel Variable Neighborhood Search Algorithm. Expert Systems with Applications 37, 678 – 687 (2010)
10. Bagheri, A., Zandieh, M., Mahdavi, I., Yazdani, M.: An Artificial Immune Algorithm for the Flexible Job-shop Scheduling Problem. Future Generation Computer Systems 26, 533 – 541 (2010)
11. Xia, W., Wu, Z.: An Effective Hybrid Optimization Approach for Multi-objective Flexible Job-shop Scheduling Problems. Computers & Industrial Engineering 48, 409 – 425 (2005)
12. Gao, J., Gen, M., Sun, L., Zhao, X.: A Hybrid of Genetic Algorithm and Bottleneck Shifting for Multiobjective Flexible Job Shop Scheduling Problems. Computers & Industrial Engineering 53, 149 – 162 (2007)
13. Nowicki, E. and Smutnicki C.: A Fast Taboo Search Algorithm for the Job-Shop Problem. Management Science 42, 797 – 813 (1996)
14. Gao, J., Sun, L., Gen M.: A Hybrid Genetic and Variable Neighborhood Descent Algorithm for Flexible Job Shop Scheduling Problems. Computers & Operations Research 35, 2892 – 2907 (2008)
15. Zhang, G., Shao, X., Li, P., Gao, L.: An Effective Hybrid Particle Swarm Optimization Algorithm for Multi-objective Flexible Job-shop Scheduling Problem. Computers & Industrial Engineering 56, 1309 – 1318 (2009)
16. Xing, L.N., Chen, Y.U., Yang, K.W.: Multi-objective Flexible Job Shop Scheduling: Design and Evaluation by Simulation Modeling. Applied Soft Computing 9, 362 – 376 (2009)
17. Li, J.Q., Pan, Q.K., Liang, Y.C.: An Effective Hybrid Tabu Search Algorithm for Multi-objective Flexible Job-shop Scheduling Problems. Computers & Industrial Engineering 59, 647 – 662 (2010)
18. Xing, L.N., Chen, Y.W., Yang, K.W.: An Efficient Search Method for Multi-objective Flexible Job Shop Scheduling Problems. Journal of Intelligent Manufacturing 20, 283 – 293 (2009)
19. Moslehi, G., Mahnam, M.: A Pareto Approach to Multi-objective Flexible Job-shop Scheduling Problem using Particle Swarm Optimization and Local Search. International Journal of Production Economics 129, 14 – 22 (2011)
20. Mostaghim, T., Teich, J.R.: Strategies for Finding Local Guides in Multi-objective Particle Swarm Optimization. In: Proceedings of the IEEE Swarm Intelligence Symposium, pp. 26 – 33. (2003)
21. Goncalves, J.F., Mendes, J.J.M., Resende, M.G.C.: A Hybrid Genetic Algorithm for the Job Shop Scheduling Problem. European Journal of Operational Research 167, 77 – 95 (2005)
22. Deb, K., Pratap, A., Agarwal, S., Meyarivan: A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II. IEEE Transactions on Evolutionary Computation 6, 182 – 197 (2002)
23. Giffler, B., Thomspon, G.L.: Algorithms for Solving Production-scheduling Problems. Operations Research 8, 487 – 503 (1960)
24. Zitzler, E., Laumanns, M., Thiele, L.: SPEA2: Improving the Strength Pareto Evolutionary Algorithm for Multiobjective Optimization. In: K.C. Giannakoglu et al. (eds.) Evolutionary Methods for Design, Optimization and Control with Applications to Industrial Problems, pp. 95–100 (2002)
25. Lee, K.M., Yamakawa, T., Lee, K.M.: A Genetic Algorithm for General Machine Scheduling Problems. In: Proceedings of International Conference on Knowledge-based Intelligent Electronic Systems, pp. 60 – 66 (1998)
26. Hansen, P., Mladenovic, N.: Variable Neighborhood Search: Principles and Applications. European Journal of Operational Research 130, 449 – 467 (2001)
27. Kacem, I., Hammadi, S., Borne, P.: Pareto-optimality Approach for Flexible Job-shop Scheduling Problems: Hybridization of Evolutionary Algorithms and Fuzzy Logic. Mathematics and Computers in Simulation 60, 245 – 276 (2002)

Table 5. List of non-dominated solutions

4 × 5			8 × 8			10 × 7			10 × 10		
C_M	W_T	W_M	C_M	W_T	W_M	C_M	W_T	W_M	C_M	W_T	W_M
11	32	10	14	77	12	11	61	11	7	42	6
11	34	9	15	75	12	11	62	10	7	43	5
12	32	8	16	73	13	12	60	12	8	41	7
13	33	7	16	77	11				8	42	5
15 × 10			Mk01 (10×6)			Mk02 (10×6)			Mk03 (15×8)		
C_M	W_T	W_M	C_M	W_T	W_M	C_M	W_T	W_M	C_M	W_T	W_M
11	91	11	40	162	38	26	151	26	204	850	204
11	93	10	40	164	37	27	145	27	210	848	210
			40	167	36	27	150	26	213	844	213
			41	160	38	28	144	28	221	842	221
			41	163	37	29	143	29	222	838	222
			42	156	40	30	142	30	231	834	231
			42	158	39	31	141	31	240	832	240
			42	165	36	33	140	33	249	830	249
			43	154	40				258	828	258
			45	153	42				267	826	267
									276	824	276
									285	822	285
									294	820	294
									303	818	303
									312	816	312
									321	814	321
									330	812	330
Mk04 (15×8)			Mk05 (15×4)			Mk06 (10×15)			Mk07 (20×5)		
C_M	W_T	W_M	C_M	W_T	W_M	C_M	W_T	W_M	C_M	W_T	W_M
61	373	60	172	687	172	62	414	57	140	695	140
62	360	61	173	683	173	63	442	53	141	692	141
62	363	60	175	682	175	64	438	50	142	688	142
63	353	62	178	680	178	65	436	50	143	684	143
63	357	61	179	679	179	66	392	56	144	673	144
63	360	60	183	677	183	67	397	54	150	669	150
64	352	64	185	676	185	68	434	50	151	667	151
65	348	63	191	675	191	69	454	49	156	664	156
66	345	66	197	674	197	70	423	53	157	662	157
67	344	66	203	673	203	71	449	49	161	660	161
67	347	65	209	672	209	72	364	68	162	659	162
69	343	67				73	363	69	166	657	166
72	340	72				74	365	67	175	655	175
									187	653	187
	…						…		202	651	202
146	324	146				102	330	100	217	649	217

Table 5. List of non-dominated solutions (continued)

Mk08 (20×10)			Mk09 (20×10)			Mk10 (20×15)		
C_M	W_T	W_M	C_M	W_T	W_M	C_M	W_T	W_M
523	2524	523	307	2273	306	208	2032	205
524	2519	524	307	2279	301	208	2037	204
533	2514	533	307	2280	300	209	2024	206
542	2509	542	307	2281	299	209	2029	205
551	2504	551	308	2278	302	209	2034	204
560	2499	560	309	2263	309	210	2028	205
569	2494	569	309	2264	308	210	2045	203
578	2489	578	309	2265	307	211	1998	207
587	2484	587	309	2266	304	211	2011	205
			309	2271	303	211	2014	204
			309	2272	302	211	2020	203
			309	2273	299	212	1986	211
			310	2262	310	212	1992	209
			311	2260	310	212	1995	207
			312	2256	312	212	2001	199
			314	2255	314	213	1963	210
			315	2254	315	213	1969	209
			316	2253	316	213	1979	207
			317	2252	316	213	1981	205
			317	2253	315	213	1989	202
			320	2247	320	213	1999	199
			321	2246	321	214	1971	207
			322	2245	322	214	1973	205
			325	2244	323	214	1979	204
			326	2242	326	214	1998	200
			327	2240	327	214	2025	197
			327	2241	326	215	1957	211
			328	2239	328	215	1978	203
			332	2237	332	215	1981	202
			333	2236	333	215	1987	201
			333	2238	331	215	1993	199
			334	2235	334	215	2022	198
			335	2234	334	216	1951	212
			335	2235	333	216	1956	207
			335	2236	332	216	1967	205
			339	2232	339	216	1971	204
			340	2231	340	216	1982	201
			341	2230	340	215	1978	203
			341	2231	339	216	1991	199
			342	2229	342	216	2000	198
				…			…	
			454	2210	454	273	1850	260