Multiobjective Flexible Job Shop Scheduling Using Memetic Algorithms

Yuan Yuan and Hua Xu

***Abstract*—In this paper, we propose new memetic algorithms (MAs) for the mutiobjective flexible job shop scheduling problem (MO-FJSP) with the objectives to minimize the makespan, total workload and critical workload. The problem is addressed in a Pareto manner, which aims to search for a set of Pareto optimal solutions. First, by using well-designed chromosome encoding/decoding scheme and genetic operators, the non-dominated sorting genetic algorithm II (NSGA-II) is adapted for the MO-FJSP. Then our MAs are developed by incorporating a novel local search algorithm into the adapted NSGA-II, where some good individuals are chosen from the offspring population for local search using a selection mechanism. Furthermore, in the proposed local search, a hierarchical strategy is adopted to handle the three objectives, which mainly considers the minimization of makespan, while the concern of the other two objectives is reflected in the order of trying all the possible actions that could generate the acceptable neighbor. In the experimental studies, the influence of two alternative acceptance rules on the performance of the proposed MAs is first examined. Afterwards, the effectiveness of key components in our MAs is verified, including genetic search, local search, and the hierarchical strategy in local search. Finally, extensive comparisons are carried out with the state-of-the-art methods specially presented for the MO-FJSP on well-known benchmark instances. The results show that the proposed MAs perform much better than all the other algorithms.**

***Note to Practitioners*—The flexible job shop scheduling problem (FJSP) has important applications in textile, automobile assembly, semiconductor manufacturing, and many other industries. In the flexible job shop, a group of machines are capable for each operation, which is different from the traditional job shop environment where each operation can be processed by only a single machine. The FJSP is quite challenging, since the decisions include not only operation sequencing but also machine assignment. In the literature, the majority of studies for the FJSP are centered on optimizing the makespan. However, a single objective is deemed as insufficient for real and practical applications. Indeed, in the industry, production managers are usually concerned with more than one objective. This paper aims to simultaneously minimize the makespan, total workload, and critical workload for the FJSP, which can lead to both high throughput and load balance of machines. This problem is solved in the posterior approach, whose goal is to seek for a set of Pareto optimal solutions. We propose effective memetic algorithms (MAs) that combine a classical multiobjective evolutionary technique referred as NSGA-II with a novel problem-specific local search. To enhance the ability to deal with multiple objectives, a hierarchical strategy is used in local search which gives varying degrees of consideration to each objective. The effectiveness of the proposed MAs is well demonstrated by extensive comparisons against the existing best-performing algorithms for the considered problem. This work can be extended to those practical problems in the flexible manufacturing system. In addition, the idea to deal with objectives hierarchically can be generalized for the other production scheduling problems with multiple objectives, such as hybrid flow shop, blocking flow shop, and no-wait job shop.**

***Index Terms*—Muti-Objective, flexible job shop scheduling, memetic algorithm, non-dominated sorting genetic algorithm II (NSGA-II), local search**

The authors are with the State Key Laboratory of Intelligent Technology and Systems, Tsinghua National Laboratory for Information Science and Technology, Department of Computer Science and Technology, Tsinghua University, Beijing 100084, PR China (e-mail: yyxhdy@gmail.com, xuhua@tsinghua.edu.cn).

Corresponding author: H. Xu (e-mail: xuhua@tsinghua.edu.cn)

I. Introduction

In the field of production scheduling, the job shop scheduling problem (JSP) is one of the most important issues because of its complexity and practical applicability in real-world situations. The flexible job shop scheduling problem (FJSP) is a generalization of the classical JSP, where each operation is allowed to be processed by any machine from a given set, rather than one specified machine. Therefore, in the FJSP, assignment of each operation to an appropriate machine is also needed besides sequencing of operations on each machine. Obviously, the FJSP is more complicated than the JSP, and it has been proved that the FJSP is strongly NP-hard even if each job has at most three operations and there are two machines [1].

Over the past decades, the single-objective FJSP (SO-FJSP), generally to minimize the makespan that is the time required to complete all jobs, has been extensively studied in the literature [2]–[9]. Compared to the SO-FJSP, the research on the multiobjective FJSP (MO-FJSP) is relatively limited. However, many real-world scheduling problems usually involve simultaneous optimization of several objectives which are in conflict to some extent. Thus, the MO-FJSP may be closer to the realistic production environments and should deserve enough attention. In recent years, the MO-FJSP has captured more and more interest, and many algorithms have been proposed. These methods to solve the MO-FJSP can be roughly categorized into two types: *priori approach* and *posteriori approach*.

In the priori approach, two or more objectives are usually linear weighted and combined into a single measure. For example, given n optimization criteria $f_1, f_2, \ldots, f_n$, a single objective problem is derived with a linear weighting function $f = \sum_{i=1}^{n} w_i f_i$, where $0 \leq w_i \leq 1$, $\sum_{i=1}^{n} w_i = 1$. However, the linear weighted summation might not always represent the trade-off relation between objectives. Besides, the determination of the weight coefficient w_i for each objective is a non-trivial task. The producers may run the prior approach many times to obtain a satisfactory solution. According to the existing literature, earlier research on the MO-FJSP mainly concentrated on this approach. Xia and Wu [10] proposed a

hierarchical method using particle swarm optimization (PSO) to assign operations on machines and simulated annealing (SA) algorithm to sequence operations on each machine. Liu *et al.* [11] presented a hybrid meta-heuristic combining PSO and variable neighborhood search (VNS) to solve the MO-FJSP. Gao *et al.* [12] developed a new genetic algorithm (GA) hybridized with a bottleneck shifting procedure. Zhang *et al.* [13] combined PSO and tabu search (TS) technique to deal with the MO-FJSP, where TS was embedded into PSO as a local search. Xing *et al.* [14] designed an efficient search method for the MO-FJSP. In their paper, ten different sets of weights were used in order to collect a set of solutions for each problem instance. Li *et al.* [15] proposed a hybrid TS algorithm for the MO-FJSP, in which neighborhood structures were developed respectively for the machine assignment module and operation scheduling module.

The posteriori approach is in fact more desirable. In this approach, the solutions are compared based on the Pareto dominance relation. A solution x is said to dominate a solution y if x is not worse than y for all objective values and is better than y for at least one objective value. A solution is optimal in the Pareto sense if and only if it is not dominated by any other solution. In contrast to the prior approach, which seeks for a single optimal solution based on the aggregated objective, the posteriori approach aims to seek for the set of Pareto optimal solutions. The posteriori approach is run without the prior information, and can exhibit the trade-off between objectives through the distribution of obtained solutions in a single run. It is helpful for the producers to evaluate these solutions and make a decision. Very recently, to solve the MO-FJSP in this Pareto way has been more concerned by the researchers. Kacem *et al.* [16] proposed a Pareto approach based on the hybridization of fuzzy logic (FL) and evolutionary algorithms to solve the MO-FJSP. Ho and Tay [17] integrated a guided local search procedure into the evolutionary algorithm, and an elitism memory was also adopted to keep all non-dominated solutions that have been found. Frutos *et al.* [18] introduced a memetic algorithm (MA) based on the non-dominated sorting genetic algorithm II (NSGA-II) [19], where SA algorithm was employed as a local search process. Wang *et al.* [20] presented a multiobjective GA based on immune and entropy principle for the MO-FJSP. Moslehi and Mahnam [21] proposed a new approach hybridizing PSO and local search. In [22] and [23], a hybrid discrete artificial bee colony (ABC) algorithm and a hybrid shuffled frog-leaping algorithm (SFLA) were developed respectively by Li *et al.*, both of which used the mechanism of NSGA-II for individual evaluation. Li *et al.* [24] also proposed a hybrid Pareto-based local search embedding a VNS based self-adaptive strategy for the same problem. Wang *et al.* [25] presented an enhanced Pareto-based ABC algorithm, in which a mix of strategies to keep quality and diversity of solutions are integrated. Rahmati *et al.* [26] adapted two multiobjective evolutionary algorithms for the MO-FJSP, and several metrics of the multiobjective evaluation were introduced into the MO-FJSP literature. Rabiee *et al.* [27] carried out a similar study, four existing muti-objective evolutionary algorithms are adapted to the partial MO-FJSP by them. Xiong *et al.* [28] developed a hybrid multiobjective evolutionary approach, where a local search based on critical path was incorporated. Chiang *et al.* [29] proposed a multiobjective evolutionary algorithm which utilizes effective genetic operators and maintains population diversity carefully. In their subsequent research [30], a muti-objective MA with an embedded variable neighborhood descent (VND) procedure was also developed.

Note that all the work on the MO-FJSP mentioned above except for [11], [18] and [27] considered the makespan, total workload and critical workload as objectives. In [11], the MO-FJSP with the makespan and flowtime criteria was studied. While in [18] and [27], only the makespan and total workload objectives were involved.

As described above, the latest studies on the MO-FJSP are more focused on the posteriori approach. Despite a number of recent achievements on this subject, there still exists potential for more research. In this paper, we adopt the posteriori approach and propose new state-of-the-art memetic algorithms (MAs), which hybridize NSGA-II based genetic search with local search, for the MO-FJSP with the criteria to minimize the makespan, total workload and critical workload. Our contributions on the study of MO-FJSP may be reflected both in *algorithm design* and *experimental analysis*.

In the aspect of algorithm design, the novelty can be summarized as follows. First and foremost, a problem-specific local search operator based on critical operations is proposed for the MO-FJSP, stressing the exploitation of the problem space. In this local search, a hierarchical strategy is used to deal with the three objectives. That is, the minimization of makespan is mainly concerned, while the consideration of the other two objectives is embodied in the order of trying all the possible actions that could generate the acceptable neighbor. Second, a genetic search based on newly adapted NSGA-II is developed for the MO-FJSP through well-designed chromosome encoding, chromosome decoding, and genetic operators, which focuses on the exploration of the problem space. Third, the similar mechanism with that presented in [31] is adopted to select individuals from the population for local search, which could be beneficial to the balance between local search and genetic search in our MAs. To our knowledge, this mechanism is introduced into the research on MO-FJSP for the first time.

As for the aspect of experimental analysis, the features lie in the following. First, the contribution of key components in our MAs to the overall performance, such as local search, is examined through the experiments. Thus, unlike the existing work, we not only show the effectiveness of MAs but also disclose how each key component contributes to the whole performance of MAs. Second, in the literature on the MO-FJSP, the typical way of presenting algorithm performance is to list the non-dominated solutions found over a certain number of runs of the algorithm. Nevertheless, it seems to be a little hard to tell how well the algorithm exactly performs just in this way, which is more qualitative than quantitative. Hence, it is necessary to bring in state-of-the-art quantitative indicators for the multiobjective optimization to make the evaluation more reasonable and sound. In fact, this practice is not extraordinary for the existing research on many MO-COPs

[31]–[34]. But for the MO-FJSP, it is still far from common.

The rest of this paper is organized as follows. In the next section, the background of MO-FJSP, NSGA-II, and MAs for MO-COPs is given. In Section III, an overview of proposed MAs is shown. The implementation details of the proposed MAs, including genetic search and local search, are respectively described in Section IV and Section V. Afterwards, experimental studies are provided in Section VI. Finally, the paper is summarized in Section VIII.

II. Background

A. Formulation of the MO-FJSP

The MO-FJSP can be formulated as the following. There are a set of n independent jobs $\mathcal{J} = \{J_1, J_2, \ldots, J_n\}$ and a set of m machines $\mathcal{M} = \{M_1, M_2, \ldots, M_m\}$. A job J_i is formed by a sequence of n_i precedence constraint operations $\{O_{i,1}, O_{i,2}, \ldots, O_{i,n_i}\}$ to be performed one after another according to the given sequence. Each operation $O_{i,j}$, i.e the jth operation of job J_i, must be executed on one machine chosen from a given subset $\mathcal{M}_{i,j} \subseteq \mathcal{M}$. The processing time of the operation is machine dependent. $p_{i,j,k}$ is denoted to be the processing time of $O_{i,j}$ on machine M_k. The scheduling consists of two subproblems: the routing subproblem that assigns each operation to an appropriate machine and the sequencing subproblem that determines a sequence of operations on all the machines.

Let C_i be the completion time of job J_i. W_k is the summation of processing time of operations that are processed on machine M_k. Three objectives namely *makespan*, *total workload*, and *critical workload* are to be minimized in this paper, which are defined respectively as follows:

$$C_{\max} = \max\{C_i | i = 1, 2, \ldots, n\} \tag{1}$$

$$W_{\mathrm{T}} = \sum_{k=1}^{m} W_k \tag{2}$$

$$W_{\max} = \max\{W_k | k = 1, 2, \ldots, m\} \tag{3}$$

Moreover, the following assumptions are made in this study: all the machines are available at time 0; all the jobs are released at time 0; each machine can process only one operation at a time; each operation must be completed without interruption once it starts; the order of operations for each job is predefined and cannot be modified; the setting up time of machines and transfer time of operations are negligible.

For illustrating explicitly, a sample instance of FJSP is shown in Table I, where rows correspond to operations and columns correspond to machines. Each entry of the input table denotes the processing time of that operation on the corresponding machine. In this table, the tag "–" means that a machine cannot execute the corresponding operation.

TABLE I
Processing time table of an instance of FJSP.

Job	Operation	M_1	M_2	M_3
J_1	$O_{1,1}$	1	–	3
	$O_{1,2}$	4	2	2
J_2	$O_{2,1}$	–	2	4
	$O_{2,2}$	1	2	3
	$O_{2,3}$	–	–	2
J_3	$O_{3,1}$	4	2	3
	$O_{3,2}$	1	–	4

B. Disjunctive Graph Model

The *disjunctive graph* [35] is originally designed to represent the schedule of the JSP. Since the FJSP is an extension of the JSP, it can be easily extended to describe the schedule of the FJSP. Suppose that the disjunctive graph is represented by $G = \{V, C \cup D\}$. V denotes a set of nodes, and each node (excluding starting and ending nodes) represents an operation; C is the set of conjunctive arcs, and each of them corresponds to the precedence constraint within the same job; D is the set of disjunctive arcs, and each of them corresponds to the precedence of operations performed on the same machine. The chosen machine for each operation is labeled above its node, while the corresponding processing time is labeled below and regarded as the weight of the node. Starting and ending nodes have weight zero. If the disjunctive graph is acyclic, it corresponds to a feasible schedule for the FJSP. And the longest path from starting node to ending node is called the *critical path*, whose length denotes the makespan. Any operation on the critical path is called a *critical operation*. In the later statement, we would not distinguish between operations and nodes, schedules and disjunctive graphs.

Take the problem shown in Table I for instance, a possible schedule represented by the disjunctive graph is depicted in Fig. 1. In the disjunctive graph, there exist two critical paths $S \to O_{2,1} \to O_{2,2} \to O_{2,3} \to E$ and $S \to O_{3,1} \to O_{2,3} \to E$ with lengths both equal to 5, so the makespan of this schedule is 5. Operations $O_{2,1}$ $O_{2,2}$, $O_{2,3}$ and $O_{3,1}$ are all critical operations.

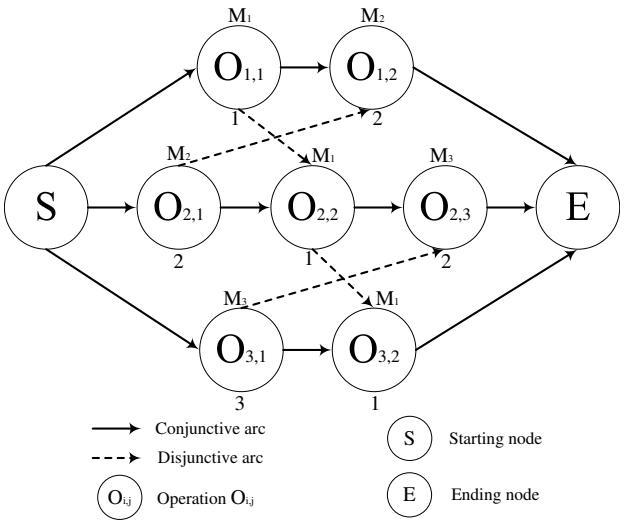

Fig. 1. Illustration of the disjunctive graph.

For the convenience of describing algorithms, we define some notations based on the disjunctive graph G. Denote $\mu(G, v)$ as the selected machine ID for a generic node v in G.

Let $ES(G, v)$ be its earliest starting time and $LS(G, v, T)$ be its latest starting time without delaying the required makespan T. Correspondingly, the earliest and latest completion time are denoted as $EC(G, v) = EC(G, v) + p_{v,\mu(G,v)}$ and $LC(G, v, T) = LS(G, v, T) + p_{v,\mu(G,v)}$ respectively, where $p_{v,\mu(G,v)}$ is the processing time of node v on machine $\mu(G, v)$. Denote $PM(G, v)$ as the operation processed on the same machine right before v and $SM(G, v)$ as the one right after v. Let $PJ(v)$ be the operation that immediately precedes v within the same job and $SJ(v)$ be the one immediately succeeds v. Let $nc(G)$ be the number of critical operations in G and $\chi(G) = \{co_1, co_2, \ldots, co_{nc(G)}\}$ be the set of critical operations.

An operation v is a critical operation, if and only if $ES(G, v) = LS(G, v, C_{\max}(G))$. In addition, because we only consider active schedules [36] in this study, the actual starting time of an operation in G is set as its earliest starting time. Take the disjunctive graph shown in Fig. 1 for instance, operation $O_{2,2}$ is a critical operation, so $ES(G, O_{2,2}) = LS(G, O_{2,2}, 5) = 2$; operation $O_{1,2}$ is not critical, $ES(G, O_{1,2}) = 2$ and $LS(G, O_{1,2}, 5) = 3$; if the required makespan $T = 6$, then $LS(G, O_{1,2}, 6) = 4$.

C. Brief Introduction to NSGA-II

The non-dominated sorting genetic algorithm II (NSGA-II) [19] is a population-based multiobjective evolutionary algorithm. The main procedure of NSGA-II can be briefly described below.

Without loss of generality, the t-th generation of NSGA-II is considered. Suppose that the parent population at this generation is P_t, while the child population is Q_t which is created by using P_t and genetic operators. The size of P_t and Q_t are both N. Thereafter, the two populations P_t and Q_t are combined together to form a new population $R_t = P_t \cup Q_t$ (of size $2N$). To choose the best N members from R_t for the next generation, a process called *non-dominated sorting* is first executed, which classifies R_t into different non-domination levels (F_1, F_2 and so on). Then the new population P_{t+1} is filled by members of different non-domination levels, one at a time. This filling starts from the best non-domination level F_1 and continues with the second one F_2, and so on. Because the overall size of R_t is $2N$, not all levels may be accommodated in P_{t+1}, and they are simply neglected. Moreover, in most situations, the last accepted level cannot be completely included. Instead of arbitrarily discarding some members from the last level, only those solutions that will maximize the diversity of selected solutions are chosen. In NSGA-II, this is achieved through using a niche strategy, which computes the *crowding distance* for each member of last level as the summation of objective-wise distance between two nearest neighbors. Then, the solutions with larger crowding distance values are selected. For further details of NSGA-II, refer to [19].

D. Memetic Algorithms for Multiobjective Combinatorial Optimization

Memetic algorithms (MAs), which hybridize evolutionary algorithms with local search, have shown high search ability in single-objective combinatorial optimization problems (SO-COPs). And a number of issues for designing these high performing MAs for the SO-COPs have also been discussed [37]–[39]. However, to develop high performing MAs for multiobjective combinatorial optimization problems (MO-COPs) is even harder, and is not so largely explored.

In MAs for the MO-COPs, an unique problem lies in the comparison of solutions in the local search. Generally, there are two schemes for coping with this problem: One uses a scalarizing function, and the other uses Pareto ranking. The research in [40] indicated that better results are obtained from the scalarizing function approach than the Pareto ranking approach. Among scalarizing functions, weighted sum of objectives is the most common one, which is first employed in the well-known multiobjective genetic local search (MOGLS) [41]. Recently, Sindhya *et al.* [42] presented a comprehensive literature review on MAs for the multiobjective optimization with different scalarizing functions adopted.

In practice, there exist many other issues that may affect the performance of MAs for the MO-COPs. Some of them are listed as follows:

1) How often should local search be applied?
2) To which solutions should local search be applied?
3) How long should local search be run?
4) Which local search procedure is to be used?

If the above issues are well addressed, it would be very likely for the designed MA to maintain good balance between exploration (population-based) and exploitation (local improvement) throughout the search, and thus to achieve good performance. In the literature, some efforts have been made on these issues. For example, Ishibuchi *et al.* [31] studied how to strike a balance between genetic search and local search in MAs for the multiobjective permutation flow shop scheduling problem (MO-PFSP). Their experimental results showed the importance of this balance. And when the balance is not appropriately specified, the performance of mutiobjective evolutionary algorithms is usually severely degraded by hybridizing with local search. Ishibuchi *et al.* [43] also examined the effect of the specification of the local search probability on the performance of MAs for the MO-PFSP and multiobjective 0/1 knapsack problem. This work suggested that dynamically changing the local search probability is better than specifying it as a constant value. In [44], the authors assumed a situation where each objective has its own powerful heuristic local search procedure. Then they proposed an idea of using such heuristic local search procedures for single-objective optimization in MOGLS. The results on the multiobjective 0/1 knapsack problem showed that this idea could improve the performance of MOGLS. Garrett and Dasgupta [45] conducted an empirical comparison of four MA strategies on the multiobjective quadratic assignment problem. These four strategies correspond roughly to "short bursts of local search on all individuals", "longer local search runs on all individuals", "short bursts of local search on randomly chosen individuals", and "longer local search runs on randomly chosen individuals".

III. Overview of the Proposed MAs

The framework of the proposed MAs is based on the original NSGA-II [19], which is depicted in Algorithm 1. First, an initial population with N chromosomes is randomly generated, where N is the population size. Then Steps 4-18 are iterated until the termination criterion is satisfied. In each generation t, the binary tournament selection, and genetic operators (crossover and mutation) are first performed to produce the offspring population Q_t. Next the local search algorithm is applied to Q_t to obtain the improved population Q'_t. In Step 6, the populations P_t, Q_t and Q'_t are merged as the population R_t. The individuals with the same objective values in R_t are eliminated by doing the mutation to the duplicates in Step 7. Finally, the best N individuals are selected as the next population P_{t+1} from R_t using fast non-dominated sorting and crowding distance.

Algorithm 1 Framework of the proposed MAs

1: $P_0 \leftarrow$ InitializePopulation()
2: $t \leftarrow 0$
3: **while** the termination criterion is not met **do**
4: $Q_t \leftarrow$ MakeOffspringPopulation(P_t)
5: $Q'_t \leftarrow$ LocalSearch(Q_t)
6: $R_t \leftarrow P_t \cup Q_t \cup Q'_t$
7: $R_t \leftarrow$ EliminateDuplicates(R_t)
8: $\{F_1, F_2, \ldots\} \leftarrow$ FastNonDominatedSort(R_t)
9: $P_{t+1} \leftarrow \emptyset$
10: $i \leftarrow 1$
11: **while** $|P_{t+1}| + |F_i| \leq N$ **do**
12: CrowdingDistanceAssignment(F_i)
13: $P_{t+1} \leftarrow P_{t+1} \cup F_i$
14: $i \leftarrow i + 1$
15: **end while**
16: Sort(F_i)
17: $P_{t+1} \leftarrow P_{t+1} \cup F_i[1 : (N - |P_{t+1}|)]$
18: $t \leftarrow t + 1$
19: **end while**

From Algorithm 1, it can be seen that the implementation of the proposed MAs concerns two crucial procedures. One is how to produce the offspring population using genetic search, corresponding to Step 4 in Algorithm 1. The other is how to generate the improved population using local search to the offspring population, corresponding Step 5 in Algorithm 1.

IV. Exploration using Genetic Search

In this section, we will detail the implementation of genetic search in the proposed MAs, including chromosome encoding, chromosome decoding and genetic operators.

A. Chromosome Encoding

A solution of the FJSP can be described by the assignment of operations to machines and the processing sequencing of operations on the machines. Therefore, a chromosome in the proposed MAs consists of two vectors: machine assignment vector and operation sequence vector, corresponding well to two subproblems in the FJSP.

Before explaining the two vectors, we first consecutively give a fixed ID for each operation in the form of j, where $j = 1, 2, \ldots, d$ with $d = \sum_{i=1}^{n} n_i$. This means that the operations $1, \ldots, n_1$ belong to job J_1, $n_1 + 1, \ldots, n_1 + n_2$ belong to J_2 and so on. After numbered, an operation can also be referred to by the fixed ID, for example, in Table I, operation 4 has the same reference with operation $O_{2,2}$.

The machine assignment vector, which is denoted by $\mathbf{u} = [u_1, u_2, \ldots, u_d]$, is an array of d integer values, where $1 \leq u_j \leq l_j$, $j = 1, 2, \ldots, d$, l_j is the size of alternative machine set for operation j. Let us sort available machines of operation j in the non-decreasing order of the time they need to execute operation j. If the same processing time is required, the machine with smaller ID ranks ahead. Then, u_j means that operation j chooses the u_jth one in its sorted available machines.

The operation sequence vector, $\mathbf{v} = [v_1, v_2, \ldots, v_d]$, is an ID permutation of all the operations. The order of occurrence for each operation in the $\mathbf{v}$ indicates its scheduling priority. For example, a possible operation sequence vector for the problem shown in Table I is represented as $\mathbf{v} = [6, 1, 7, 3, 4, 2, 5]$. And it can be directly translated into a unique list of ordered operations: $O_{3,1} \succ O_{1,1} \succ O_{3,2} \succ O_{2,1} \succ O_{2,2} \succ O_{1,2} \succ O_{2,3}$. Operation $O_{3,1}$ has the highest priority and is scheduled first, then operation $O_{1,1}$, and so on. It must be noted that not all the ID permutations are feasible for the operation sequence vector because of the designated priority of operations lying in a job. That is to say, the operations within a job should keep the relative priority order in the $\mathbf{v}$.

B. Chromosome Decoding

The decoding of the chromosome is to allocate a period of time for each operation on its assigned machine one by one according to their order in the $\mathbf{v}$. When one operation is treated, its selected machine is first got from the $\mathbf{u}$, then the idle time intervals between operations that have already been scheduled on that machine are scanned from left to right until an available one is found. Let $s_{i,j}$ be the stating time of a generic operation $O_{i,j}$ in the schedule and $c_{i,j}$ its completion time. Since an operation can only be started after the completion of its immediate precedent operation within the same job, the idle time interval $[S_x, E_x]$ on machine M_k is available for $O_{i,j}$, if

$$\begin{cases} \max\{S_x, c_{i,j-1}\} + p_{i,j,k} \leq E_x, & \text{if } j \geq 2 \\ S_x + p_{i,j,k} \leq E_x, & \text{if } j = 1 \end{cases} \tag{4}$$

When $O_{i,j}$ is to be allocated in the available interval $[S_x, E_x]$, $\max\{S_x, c_{i,j-1}\}$ ($j \geq 2$) or S_x ($j = 1$) is taken as its starting time. If no such interval exists on machine M_k for $O_{i,j}$, it would be arranged at the end of M_k. A schedule generated by using this decoding method can be ensured to be an active schedule [46]. For example, a chromosome for the problem shown in Table I, such as $\mathbf{u} = [1, 1, 1, 1, 1, 2, 1]$ and $\mathbf{v} = [6, 1, 7, 3, 4, 2, 5]$, is just decoded into an active schedule represented by the Gantt chart which is illustrated in Fig. 2.

In this decoding method, an operation is allowed to search the earliest available idle time interval on the assigned machine when scheduled. Hence, for two operations v_i and v_j assigned on the same machine, v_i may be stared earlier than v_j in the decoded schedule, while v_j actually appears before v_i in the

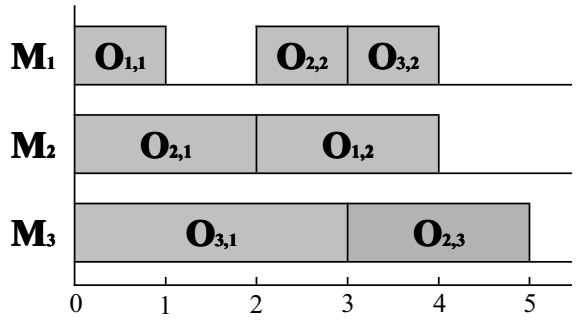

Fig. 2. Gantt chart corresponding to the chromosome.

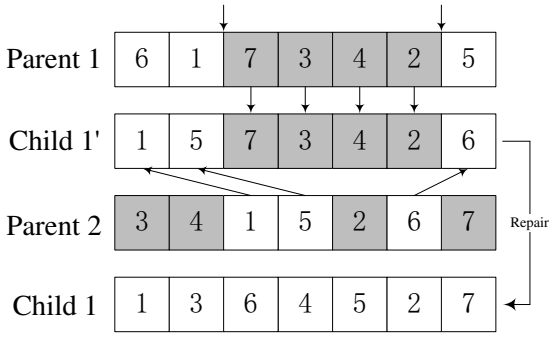

Fig. 3. Illustration of the crossover for the $\mathbf{v}$.

$\mathbf{v}$. To make the operation sequence information well inherited, when a chromosome is decoded, the operations in its $\mathbf{v}$ are reordered according to their starting time in the corresponding decoded schedule before it involves genetic operators.

C. Genetic Operators

The genetic operators in our MAs include crossover and mutation, which are conducted to produce offsprings. The crossover is applied to a pair of chromosomes, while the mutation is applied to a single individual.

The crossover operators for two vectors in the chromosome are implemented respectively. For the $\mathbf{u}$, a subset of positions are first randomly chosen, then generate the $\mathbf{u}$ of children by exchanging the values of these selected positions between two parents. As for the $\mathbf{v}$, a modified order crossover [47] is used. It can be described as follows. First, two points are randomly picked, and operations between the two points in the first parent are selected. Then, copy these operations to the corresponding positions in the first child. Finally, complete this child with the remaining operations, in the same priority order they appear in the second parent. However, the obtained operation sequence may be not feasible, due to the constraints between operations within a job. So, a simple repair procedure presented in Algorithm 2 is further executed to adjust the relative order of operations in the same job. In Fig. 3, the above crossover operator is illustrated for the problem shown in Table I. The symmetric process is repeated for the second parent and the second child.

Algorithm 2 RepairOperationSequence ($\mathbf{v}$)

```
[q_1, q_2, ..., q_n] ← [0, 0, ..., 0]
for i = 1 to d do
    Get the job J_k that the operation v_i belongs to
    q_k ← q_k + 1
    Get the fixed ID op for the operation O_{k,q_k}
    v_i ← op
end for
```

The mutation also consists of two parts. For the $\mathbf{u}$, it is achieved by changing the machine assignment of a single operation that is chosen arbitrarily. With regards to the $\mathbf{v}$, the mutation is done by inserting an operation to another position in the $\mathbf{v}$ without violating the designated priority among operations of the same job, where both the operation and the position are randomly selected.

V. Exploitation using Local Search

In this section, we will describe in detail how to do the local search to the offspring population. There exist two parts for the implementation. The first is the selection of individuals from the offspring population for local search. The second is that, once a chromosome is selected, we should further consider how to refine it via local search in order to obtain several improved ones, which are to be added into the improved population.

A. Selection of Individuals for Local Search

In our MAs, a selection mechanism similar to that presented in [31] is adopted. First, the following aggregation function is defined:

$$f(\mathbf{x}, \boldsymbol{\lambda}) = \lambda_1 f_1(\mathbf{x}) + \lambda_2 f_2(\mathbf{x}) + \lambda_3 f_3(\mathbf{x}) \tag{5}$$

where $\boldsymbol{\lambda} = [\lambda_1, \lambda_2, \lambda_3]$ is a weight vector, $f_1(\mathbf{x})$, $f_2(\mathbf{x})$ and $f_3(\mathbf{x})$ are directly set to three objectives of the MO-FJSP defined in (1), (2) and (3) respectively. We generate a set of uniformly distributed weight vectors, which meet the following condition:

$$\lambda_1 + \lambda_2 + \lambda_3 = z, \quad \lambda_i \in \{0, 1, \ldots, z\},\ i = 1, 2, 3 \tag{6}$$

Therefore, the number of the weight vectors is calculated as C_{z+2}^2. In this paper, z in (6) is set as 23 to produce 300 vectors for three objective problems. When selecting an individual for local search, a weight vector is randomly drawn from the weight vector set. Next, an individual is chosen from the current population using tournament selection with replacement based on (5) with the current weight vector. Finally, the local search is applied to the chosen individual to obtain a set E_i that contains a certain number of improved solutions.

Another question is how often should local search be applied. We introduce a local search probability P_{ls} and the number of $\lfloor N \times P_{ls} \rfloor$ individuals are selected for local search. In other words, the selection of an individual and the applicability of local search are repeated $\lfloor N \times P_{ls} \rfloor$ times. All the above-mentioned procedures are summarized in Algorithm 3.

Algorithm 3 LocalSearch(Q_t)

1: $Q'_t \leftarrow \emptyset$
2: **for** $i = 1$ to $\lfloor N \times P_{ls} \rfloor$ **do**
3: Randomly draw a weight vector $\boldsymbol{\lambda}$ from the weight vector set
4: $\{\mathbf{u}, \mathbf{v}\} \leftarrow$ TournamentSelectionWithReplacement($Q_t, \boldsymbol{\lambda}$)
5: $E_i \leftarrow$ LocalSearchForIndividual($\{\mathbf{u}, \mathbf{v}\}, \boldsymbol{\lambda}$)
6: $Q'_t \leftarrow Q'_t \cup E_i$
7: **end for**
8: **return** Q'_t

B. Local Search to a Chromosome

So far, there is one last key issue remaining to be addressed. That is the local search operator to a chromosome corresponding to Step 5 in Algorithm 3. In our MAs, the proposed local search is not directly applied to a chromosome, but in fact to the decoded schedule of the chromosome, which is helpful for introducing the problem-specific knowledge.

1) Neighbor Generation in Local Search: In our local search, the neighbor of a schedule G is obtained by moving an operation. Since the objective makespan is relatively harder to be minimized, we accept the neighbor G' of G if and only if $C_{\max}(G') \leq C_{\max}(G)$. To make the move profitable, we introduce the following theorem

Theorem 1: In the schedule G, if a new schedule G' is obtained by moving an operation $O_{i,j} \notin \chi(G)$ in G, then $C_{\max}(G) \leq C_{\max}(G')$.

Proof: Suppose that P be a critical path in G, which is represented as $S \rightarrow co_1 \rightarrow \ldots \rightarrow co_l \rightarrow E$. Because $O_{i,j} \notin \chi(G)$, $O_{i,j}$ is not on P. The move of $O_{i,j}$ has the following three possible situations: (1) move to the position between two operations on one machine; (2) move to the position ahead of all the operations on one machine; (3) move to the position behind all the operations on one machine. For the first situation, we assume that $O_{i,j}$ is moved to the position between operations o_x and o_y on one machine. It is obvious that if there does not exist $k \in \{1, 2, \ldots, l-1\}$, $co_k = o_x$ and $co_{k+1} = o_y$, then P would not be influenced and still exists in G'. Hence, $C_{\max}(G) \leq C_{\max}(G')$. Otherwise, the path $S \rightarrow co_1 \rightarrow \ldots co_k \rightarrow O_{i,j} \rightarrow co_{k+1} \ldots \rightarrow co_l \rightarrow E$ would exist in G', and $C_{\max}(G) < C_{\max}(G')$. Taken together, $C_{\max}(G) \leq C_{\max}(G')$. As for the other two situations, the proof is similar. ■

From Theorem 1, the makespan can only be improved by moving critical operations, so only the critical operations are considered to be moved in our local search.

Next, we discuss how to move a critical operation in G as a result that the obtained neighbor schedule G' is feasible and $C_{\max}(G') \leq C_{\max}(G)$. Suppose a critical operation co_i in G is to be moved, first delete it from G to yield G_i^- by removing the disjunctive arc from co_i and the disjunctive arc to co_i, connecting $PM(G, co_i)$ to $SM(G, co_i)$ with a disjunctive arc, and set the weight of node co_i as 0. Then, co_i is inserted into another feasible position in G_i^- to obtain G' so that $C_{\max}(G') \leq C_{\max}(G)$. If such a position locate before operation v on machine M_k in G_i^-, co_i should be started as early as $EC(G_i^-, PM(G_i^-, v))$, and can be completed as late as $LS(G_i^-, v, C_{\max}(G))$ without delaying $C_{\max}(G)$. Besides, co_i must follow the precedence constraints within the same job. Hence, if this position before v is said to be available for co_i to insert into, the following inequality should be satisfied

$$\max\{EC(G_i^-, PM(G_i^-, v)), EC(G_i^-, PJ(co_i))\} + p_{co_i,k} \leq \min\{LS(G_i^-, v, C_{\max}(G)), LS(G_i^-, SJ(co_i), C_{\max}(G))\} \tag{7}$$

But, in fact, the "$<$" not the "$\leq$" is used in (7) in our local search due to the following theorem

Theorem 2: The schedule G' is obtained by inserting an operation co_i into a position located before operation v on machine M_k in G_i^- under (7) without equality. If $C_{\max}(G') = C_{\max}(G)$, then co_i is not the critical operation in G'.

Proof: First, according to the definition, we have

$$ES(G', co_i) = \max\{EC(G', PM(G_i^-, v)), EC(G', PJ(co_i))\} \tag{8}$$

$$LS(G', co_i, C_{\max}(G')) = \min\{LS(G', v, C_{\max}(G')), LS(G', SJ(co_i), C_{\max}(G'))\} - p_{co_i,k} \tag{9}$$

The insertion of co_i would not change the earliest completion time of $PM(G_i^-, v)$ and $PJ(co_i)$, and the latest starting time of v and $SJ(co_i)$ without delaying $C_{\max}(G)$, so we have

$$EC(G', PM(G_i^-, v)) = EC(G_i^-, PM(G_i^-, v)) \tag{10}$$

$$EC(G', PJ(co_i)) = EC(G_i^-, PJ(co_i)) \tag{11}$$

$$LS(G', v, C_{\max}(G)) = LS(G_i^-, v, C_{\max}(G)) \tag{12}$$

$$LS(G', SJ(co_i), C_{\max}(G)) = LS(G_i^-, SJ(co_i), C_{\max}(G)) \tag{13}$$

Since $C_{\max}(G') = C_{\max}(G)$, we have

$$ES(G', co_i) \neq LS(G', co_i, C_{\max}(G')) \tag{14}$$

based on (8), (9), (10), (11), (12) and (13). Therefore, co_i is not the critical operation in G'. ■

According to Theorem 2, if $C_{\max}(G') = C_{\max}(G)$, then co_i is not the critical operation in G'. Hence, it is guaranteed that co_i would not be moved in G' in the next local iteration to come back to G, which is helpful to avoid cyclic search as much as possible.

However, to insert co_i before v under (7) is met can not ensure the yielded G' is acyclic. Below, the positions to be examined on machine M_k are restricted to only feasible ones. Let Θ_k be the set of operations processed by machine M_k in G_i^- and ordered with the increasing of the earliest starting time (note that $co_i \notin \Theta_k$). Denote Φ_k and Ψ_k as two subsequences of Θ_k and are defined as follows

$$\Phi_k = \{r \in \Theta_k | ES(G, r) + p_{r,k} > ES(G_i^-, co_i)\} \tag{15}$$

$$\Psi_k = \{r \in \Theta_k | LS(G, r, C_{\max}(G)) < LS(G_i^-, co_i, C_{\max}(G))\} \tag{16}$$

Let Υ_k be the set of positions before all the operations of $\Phi_k \setminus \Psi_k$ and after all the operations of $\Psi_k \setminus \Phi_k$. Then the following theorem holds

Theorem 3: In G_i^-, the schedule obtained by inserting co_i into a position $\gamma \in \Upsilon_k$ is always feasible, and there exists a position in the set Υ_k so that no better makespan can be got by inserting co_i into any other positions on machine M_k.

The proof line of this theorem can be referred in [3]. According to Theorem 3, we have the direct corollary as follows

Corollary 1: In G_i^-, if a feasible schedule G' satisfying $C_{\max}(G') \leq C_{\max}(G)$ can be obtained by inserting co_i into a position on machine M_k, then there always exists such a schedule that is yielded by inserting co_i into a position in the set Υ_k.

With Corollary 1, only the positions in Υ_k are checked when reallocating a position on machine M_k for co_i in G_i^-, once a position satisfying (7) is found, co_i is inserted immediately and an acceptable neighbor schedule G' is formed to replace the current schedule G. As can be seen, the first move strategy is adopted in our local search where local search accepts the first acceptable neighbor, because it is especially computational expensive to check all the positions for all the critical operations. Denote $co_i \rightsquigarrow M_k$ as the action to find a position on machine M_k for co_i to insert into in G_i^-. In summarize, the action $co_i \rightsquigarrow M_k$ is depicted in Algorithm 4.

Algorithm 4 InsertOperationOnMachine(G_i^-, co_i, k)

```
Get the set of positions Υ_k on machine M_k
for each position γ in Υ_k do
    if γ satisfies (7) then
        Insert co_i into γ in G_i^-
        return true
    end if
end for
return false
```

Algorithm 5 GetNeighborSchedule(G)

```
Get the set χ(G) = {co_1, co_2, ..., co_nc}
[G_1^-, G_2^-, ..., G_nc^-] ← [∅, ∅, ..., ∅]
Sort φ(G) according to Δ_t and Δ_c
for each action co_i ⇝ M_k in the sorted φ(G) do
    if G_i^- = ∅ then
        Clone a copy of G to G*
        Delete co_i from G*
        G_i^- ← G*
    end if
    if InsertOperationOnMachine(G_i^-, co_i, k) then
        G' ← G_i^-
        return G'
    end if
end for
return ∅
```

Let $\varphi(G) = \{co_i \rightsquigarrow M_k | i = 1, 2, \ldots, nc, M_k \in \mathcal{M}_{co_i}\}$, which consists of all $\sum_{i=1}^{nc} l_{co_i}$ possible actions described in Algorithm 4 to form an acceptable neighbor G' of G. The concern of total workload and critical workload in our local search is reflected in the order of trying these actions. We define two metrics for each action $co_i \rightsquigarrow M_k$ in $\varphi(G)$ as follows

$$\Delta_t(co_i \rightsquigarrow M_k) = p_{co_i,k} - p_{co_i,\mu(co_i,G)} \tag{17}$$

$$\Delta_c(co_i \rightsquigarrow M_k) = W_k(G) + p_{co_i,k} \tag{18}$$

It is clear that the metrics Δ_t and Δ_c consider the objectives total workload and critical workload respectively. Based on the two metrics, the actions in $\varphi(G)$ are sorted according to the non-decreasing order of Δ_t. If two actions have the same Δ_t values, the one with lower Δ_c value first. At one iteration of local search, these actions are considered one after another in the sorted order until an acceptable neighbor of G is obtained. This procedure is illustrated in Algorithm 5.

From the above, a hierarchical strategy is in fact used in our local search to deal with the three objectives, which can be summed up as follows. First, we design the neighborhood structure to ensure that the makespan is non-increasing during a local iteration. In this premise, the neighbor with the smallest total workload is chosen as the new schedule at each local iteration. If there exist more than one such neighbor, the critical workload is further considered.

2) Acceptance Rules in Local Search: Now, it is relatively easy to form the procedure of local search to a chromosome which is indeed an iterative process of Algorithm 5. The only question is that what solutions that generated in the path of local search should be accepted as the improved ones and added into the improved population. Two different acceptance rules are investigated in our local search, which are referred as "Best" and "Pareto" respectively in this paper. The acceptance rule "Best" is to choose the schedule with the lowest $f(G, \boldsymbol{\lambda})$ value in the path of local search as the improved solution. In Algorithm 6, the local search operator to a chromosome using the acceptance rule "Best" is summarized, where the parameter $iter_{\max}$ is the maximal iterations of local search. As mentioned before, the proposed local search is not directly applied to a chromosome, but in fact to the decoded schedule of the chromosome, so the decoding procedure should be executed in Step 2 before the local iterations. And after the local iterations have been finished, the accepted schedule needs to be encoded in Step 14 and returned in the form of chromosome. As for the acceptance rule "Pareto", it means that, at each generation, the non-dominated ones among the solutions located in all the local search paths for local search individuals make up the improved population.

Algorithm 6 LocalSearchForIndividual($\{\mathbf{u}, \mathbf{v}\}$, $\boldsymbol{\lambda}$)

```
1: i ← 0
2: G ← ChromosomeDecoding(u, v)
3: G_best ← G
4: flag ← 0
5: while G ≠ ∅ and i < iter_max do
6:     G ← GetNeighborSchedule(G)
7:     if G ≠ ∅ and f(G, λ) < f(G_best, λ) then
8:         G_best ← G
9:         flag ← 1
10:    end if
11:    i ← i + 1
12: end while
13: if flag = 1 then
14:    {u', v'} ← ChromosomeEncoding(G_best)
15:    return {u', v'}
16: end if
17: return ∅
```

VI. Experimental Studies

The proposed algorithms are all implemented in Java language and run on an Intel Core i7-3520M 2.9GHz processor

with 8Gb of RAM. The algorithms are tested on four sets of well-known benchmark instances including 5 Kacem instances (ka4x5, ka08, ka10x7, ka10x10, ka15x10) [16], 10 BRdata instances (Mk01-Mk10) [2], 18 DPdata instances (01a-18a) [48], and 3 Hurink Vdata instances (la30, la35, la40) [49]. These sets cover almost all the problem instances ever adopted in the literature on MO-FJSP. In fact, most existing studies considered only a part of these instances. However, in our experiments, all the 36 problem instances will be used so as to make a comprehensive evaluation of all the implemented algorithms.

Because of the different strategies employed in the global search phase and local search phase, several variants of the implemented algorithms will be involved in the experiments, which are listed in Table II. MA-2 is different from MA-1 only in that it uses the "Pareto" acceptance rule in local search instead of the "Best". MA-1-NH and MA-2-NH correspond to MA-1 and MA-2 respectively, and the only difference is that neither of them uses the hierarchical strategy in local search. In other words, when generating the acceptable neighbor in local search, MA-1-NH (MA-2-NH) exams the actions in the set $\varphi(G)$ in the original order until an acceptable neighbor is obtained. To verify the effectiveness of the genetic search, the multi-start random local search (MRLS) algorithms are also designed. Specifically, MRLS-1 (MRLS-2) is formed by replacing the genetic operators in MA-1 (MA-2) with random generating method to produce new solutions. The NSGA-II here is an adapted version for the MO-FJSP, in which the chromosome representation and genetic operators of the proposed MAs are directly used. It is a pure genetic-based search algorithm without local search.

TABLE II
VARIANTS OF THE IMPLEMENTED ALGORITHMS.

Variant	Global Search	Local Search	
		Acceptance Rule	Hierarchical Strategy
MA-1	Genetic	Best	Yes
MA-2	Genetic	Pareto	Yes
MA-1-NH	Genetic	Best	No
MA-2-NH	Genetic	Pareto	No
MRLS-1	Random	Best	Yes
MRLS-2	Random	Pareto	Yes
NSGA-II	Genetic	–	–

The parameter settings of the implemented algorithms are listed in Table III. We adopt the uniform parameter values for these algorithms. Each algorithm will be terminated when the predefined number of examined solutions is exhausted. This limit is set to 150,000, 500,000, and 1,000,000 for the Kacem instances, BRdata instances and the remaining instances respectively, and is the same for all the implemented algorithms to ensure a fair comparison. For each problem instance, all the algorithms listed in Table II are independently run 30 times.

A. Performance Metrics

In order to evaluate the performance of the proposed algorithms, the inverted generational distance (IGD) [50] and the set coverage [51] are used as indicators in our experiments. They can be expressed as follows:

TABLE III
PARAMETER SETTINGS OF THE IMPLEMENTED ALGORITHMS.

Parameter	Value
Population size (N)	300
Crossover probability (P_c)	1.0
Mutation probability (P_m)	0.1
Local search probability (P_{ls})	0.1
Maximal iterations of local search ($iter_{\max}$)	50
Tournament size for local search solution selection (S_t)	20

1) **Inverted Generational Distance**: Let P^* be a set of uniformly distributed points along the Pareto front (PF). Let A be an approximation to the PF. The metric IGD of the set A is defined as:

$$\text{IGD}(A, P^*) = \frac{1}{|P^*|} \sum_{x \in P^*} \min_{y \in A} d(x, y) \tag{19}$$

where $d(x, y)$ is the Euclidean distance between the points x and y. If $|P^*|$ is large enough to represent the PF, the IGD(A, P^*) could measure both the diversity and convergence of A in a sense. To have a small value of IGD(A, P^*), A must be very close to P^* and cannot miss any part of P^*.

In this study, the metric IGD is computed based on the normalized objective vectors of the non-dominated solutions, which could be obtained by

$$\tilde{f}_i(\mathbf{x}) = (f_i(\mathbf{x}) - f_i^{\min})/(f_i^{\max} - f_i^{\min}), \;\; i = 1, 2, 3 \tag{20}$$

where $f_i^{\max}$ and $f_i^{\min}$ are the maximal and minimal values of $f_i(x)$ among all the results obtained over all runs by all the compared algorithms for the MO-FJSP.

2) **Set Coverage**: Let A and B be two approximations to the PF, the set coverage $C(A, B)$ represents the percentage of solutions in B that are dominated by at least one solution in A, i.e.

$$C(A, B) = \frac{|\{x \in B | \exists y \in A : y \text{ dominates } x\}|}{|B|} \tag{21}$$

$C(B, A)$ is not necessarily equal to $1 - C(A, B)$. If $C(A, B)$ is large and $C(B, A)$ is small, then A is better than B in a sense.

Since the actual PFs for test instances are not known, P^* is mainly formed for each problem instance by gathering all non-dominated solutions found by all the implemented algorithms in all runs. In addition, the non-dominated solutions obtained by the algorithms in the literature [23], [24], [29], [30] are also included. These P^* sets together with the detail computational results in this paper are available on the website[1] for future use of other researchers.

B. Investigation of the Acceptance Rules in Local Search

To investigate the impact of two different acceptance rules ("Best" and "Pareto") in local search on the proposed MAs, the performance comparison between MA-1 and MA-2 is carried out in this subsection.

[1]http://learn.tsinghua.edu.cn:8080/2012310563/TASE-MOFJSP-Results.rar

TABLE IV
COMPARISON OF TWO DIFFERENT ACCEPTANCE RULES IN LOCAL SEARCH ON AVERAGE IGD AND SET COVERAGE VALUES OVER 30 INDEPENDENT RUNS FOR ALL 36 PROBLEM INSTANCES.

Instance	$n \times m$	Flex.	IGD		MA-1 (A) vs MA-2 (B)	
			MA-1	MA-2	$C(A,B)$	$C(B,A)$
ka4x5	4×5	5	0.000000	0.000000	0.000000	0.000000
ka08	8×8	6.48	0.000000	0.000000	0.000000	0.000000
ka10x7	10×7	7	0.000000	0.000000	0.000000	0.000000
ka10x10	10×10	10	**0.033793**	0.092931	0.000000	0.000000
ka15x10	15×10	10	0.002778	0.000000	0.000000	0.044444
Mk01	10×6	2.09	0.001569	0.001774	0.035455	0.023333
Mk02	10×6	4.1	0.009685	**0.006263**	0.052315	**0.131944**
Mk03	15×8	3.01	0.000000	0.000000	0.000000	0.000000
Mk04	15×8	1.91	0.005937	0.006307	0.143948	0.122322
Mk05	15×4	1.71	0.003265	0.005062	0.003030	0.003030
Mk06	10×15	3.27	0.039592	0.040446	0.252925	**0.499238**
Mk07	20×5	2.83	0.000000	0.000181	0.006250	0.000000
Mk08	20×10	1.43	0.000000	0.000000	0.000000	0.000000
Mk09	20×10	2.52	0.002534	**0.001927**	0.110502	**0.220580**
Mk10	20×15	2.98	0.032299	**0.026703**	0.229056	**0.663331**
01a	10×5	1.13	0.142513	0.121680	0.366667	0.550000
02a	10×5	1.69	0.029790	0.034413	0.527354	0.364881
03a	10×5	2.56	**0.022955**	0.029485	0.582381	0.375278
04a	10×5	1.13	0.017067	0.020658	0.413626	0.282165
05a	10×5	1.69	0.029702	**0.027830**	0.373163	**0.525628**
06a	10×5	2.56	0.020006	0.020205	0.517712	0.410313
07a	15×8	1.24	0.138625	**0.104688**	0.376984	0.544444
08a	15×8	2.42	**0.025773**	0.038654	**0.602632**	0.282778
09a	15×8	4.03	0.035028	**0.026466**	0.266865	**0.613413**
10a	15×8	1.24	0.040129	0.036169	0.349751	0.606132
11a	15×8	2.42	0.024052	**0.020052**	0.285550	**0.603962**
12a	15×8	4.03	0.017390	**0.015395**	0.349964	**0.534587**
13a	20×10	1.34	0.058361	0.061297	**0.581839**	0.269497
14a	20×10	2.99	0.041484	**0.030463**	0.183185	**0.732829**
15a	20×10	5.02	0.036633	**0.021958**	0.161310	**0.809418**
16a	20×10	1.34	**0.043449**	0.047622	0.383714	0.498220
17a	20×10	2.99	0.022545	**0.015152**	0.142421	**0.742204**
18a	20×10	5.02	0.019398	**0.015005**	0.298604	**0.582840**
la30	20×10	4.65	0.040453	**0.035973**	0.196376	**0.700000**
la35	30×10	4.65	0.016722	**0.012918**	0.169444	**0.613889**
la40	15×15	6.48	0.028966	**0.016913**	0.181508	**0.717222**

For each instance and each metric, the result that is significantly better than the other is marked in bold (with smaller IGD, with greater set coverage).

In Table IV, average IGD and set coverage values over 30 independent runs of MA-1, MA-2 on all 36 problem instances are presented. The characteristics of the instances are also provided. The first column symbolizes the name for each instance; the second column shows the size of the instance, in which n stands for the number of jobs and m represents the number of machines; the third column lists the flexibility of each instance, which means the average number of alternative machines for each operation in the problem. For each instance and each performance metric (IGD and set coverage), the Wilcoxon signed-rank test [52] is further carried out on the results obtained by 30 runs of the compared algorithms, and the one that is significantly better than that of the others (with the significance level of 0.05) is marked in bold. In the following Tables V, VI and VII, the number in bold would indicate the same meaning.

From Table IV, MA-1 is significantly better than MA-2 on the instance ka10x10 and 3 DPdata instances. MA-2 significantly outperforms MA-1 on 15 out of total 36 instances, including 3 BRdata instances, 9 DPdata instances, and all 3 Hurink Vdata instances. There are remaining 17 problem instances on which neither of MA-1 and MA-2 performs significantly better than the other. As for the set coverage metric, MA-1 is significantly better than MA-2 on the instances 08a and 13a. MA-2 performs significantly better than MA-1 on 4 BRdata instances, 8 DPdata instances, and all 3 Hurink Vdata instances. There are all 5 Kacem instances, 6 BRdata instances, 8 DPdata instances on which the results have no statistical significance.

As a whole, it seems that MA-2 performs better on the considered problem instances. However, MA-1 still exhibits a little stronger search ability than MA-2 on several instances. In addition, there exists no significant advantage for MA-1 or MA-2 on almost half of all the instances. In conclusion, the "Pareto" acceptance rule is preferred for the proposed MAs according to the computational results, but the "Best" acceptance rule may be more suitable in certain cases.

C. Effect of Hybridizing Genetic Search and Local Search

In this subsection, the experiments and comparisons are conducted between MA-1 (MA-2) with NSGA-II and MRLS-1 (MRLS-2) algorithms to show the effectiveness of hybridizing genetic-based global search and problem-specific local search. We would like to gain a better understanding of why the proposed MAs work through these experiments.

In Table V, average IGD values over 30 independent runs for all 36 problem instances are reported. Four algorithm variants are divided into two groups: {MA-1, MRLS-1, NSGA-II} and {MA-2, MRLS-2, NSGA-II} for the comparing. For the first group, it is clear that MA-1 is the winner since the results of IGD generated by MA-1 are significantly better than those by MRLS-1 and NSGA-II on overwhelming majority of all instances. And there exists no statistical significance between the three algorithms on the rest instances. As for the second group, the situation is similar to the first one. it is easily observed that MA-2 significantly outperforms MRLS-2 and NSGA-II on 31 out of total 36 instances in terms of IGD metric. Neither of MRLS-2 and NSGA-II is significantly better than the other two algorithms on any instance. Table VI shows the results of set coverage by comparing MA-1 (MA-2) with MRLS-1 (MRLS-2) and NSGA-II. From Table VI, it is obvious that the results obtained by MA-1 (MA-2) are much better than those by MRLS-1 (MRLS-2) and NSGA-II.

Based on the above results and comparisons, it is concluded that MA-1 (MA-2) is much more powerful than MRLS-1 (MRLS-2) and NSGA-II, which well demonstrates the effectiveness of genetic search, local search, and the hybridization. The success of our MAs could be attributed that MA-1 (MA-2) integrates the advantages of genetic search for diversification and local search for intensification, which well achieves the balance between exploration and exploitation.

D. Effect of the Hierarchical Strategy in Local Search

In this subsection, we would demonstrate the effectiveness of the hierarchical strategy used in local search. Thus, the comparison between MA-1 (MA-2) and MA-1-NH (MA-2-NH) is carried out. It can be observed from Tables VII that

TABLE V
PERFORMANCE EVALUATION OF THE EFFECT OF HYBRIDIZING GENETIC SEARCH AND LOCAL SEARCH USING AVERAGE IGD VALUES OVER 30 INDEPENDENT RUNS FOR ALL 36 PROBLEM INSTANCES.

Instance	IGD			IGD		
	MA-1	MRLS-1	NSGA-II	MA-2	MRLS-2	NSGA-II
ka4x5	0.000000	0.047126	0.000000	0.000000	0.004167	0.000000
ka08	0.000000	0.058269	0.000833	0.000000	0.009167	0.000833
ka10x7	0.000000	0.000000	0.007407	0.000000	0.000000	0.007407
ka10x10	0.033793	0.081590	0.134327	0.092931	0.052079	0.134327
ka15x10	**0.002778**	0.313175	0.280442	**0.000000**	0.294616	0.280442
Mk01	**0.001569**	0.101091	0.035556	**0.001774**	0.090541	0.035556
Mk02	**0.009685**	0.128757	0.050101	**0.006263**	0.119288	0.050101
Mk03	0.000000	1.007897	0.000202	0.000000	0.953574	0.000202
Mk04	**0.005937**	0.251465	0.023388	**0.006307**	0.233541	0.023388
Mk05	**0.003265**	0.243095	0.012698	**0.005062**	0.209437	0.012698
Mk06	**0.039592**	0.452603	0.073385	**0.040446**	0.453476	0.073385
Mk07	**0.000000**	0.292828	0.006943	**0.000181**	0.278908	0.006943
Mk08	0.000000	0.655307	0.000000	0.000000	0.603192	0.000000
Mk09	**0.002534**	0.462225	0.014184	**0.001927**	0.439581	0.014184
Mk10	**0.032299**	0.750063	0.071379	**0.026703**	0.723913	0.071379
01a	0.142513	0.648216	0.148463	**0.121680**	0.592304	0.148463
02a	**0.029790**	0.266288	0.059942	**0.034413**	0.248628	0.059942
03a	**0.022955**	0.260912	0.065892	**0.029485**	0.226697	0.065892
04a	**0.017067**	0.152115	0.029280	**0.020658**	0.130728	0.029280
05a	**0.029702**	0.351096	0.058806	**0.027830**	0.337028	0.058806
06a	**0.020006**	0.539079	0.086530	**0.020205**	0.510495	0.086530
07a	0.138625	0.419420	0.157773	**0.104688**	0.406506	0.157773
08a	**0.025773**	0.439693	0.112187	**0.038654**	0.425074	0.112187
09a	**0.035028**	0.408097	0.117332	**0.026466**	0.402469	0.117332
10a	**0.040129**	0.312350	0.092129	**0.036169**	0.300337	0.092129
11a	**0.024052**	0.535776	0.089533	**0.020052**	0.516628	0.089533
12a	**0.017390**	0.788284	0.113920	**0.015395**	0.767139	0.113920
13a	**0.058361**	0.528276	0.086919	**0.061297**	0.510652	0.086919
14a	**0.041484**	0.509893	0.140580	**0.030463**	0.491917	0.140580
15a	**0.036633**	0.509471	0.111058	**0.021958**	0.493920	0.111058
16a	**0.043449**	0.439592	0.102748	**0.047622**	0.422147	0.102748
17a	**0.022545**	0.707046	0.132775	**0.015152**	0.701669	0.132775
18a	**0.019398**	0.942038	0.187137	**0.015005**	0.919420	0.187137
la30	**0.040453**	0.332309	0.108040	**0.035973**	0.310914	0.108040
la35	**0.016722**	0.440808	0.089383	**0.012918**	0.432735	0.089383
la40	**0.028966**	0.417990	0.140409	**0.016913**	0.400358	0.140409

For each instance, the result that is significantly better than the others is marked in bold (with smallest IGD).

the hierarchical strategy in local search further improve the performance of the proposed MAs.

Specifically, for the IGD metric, MA-1 is significantly better than MA-1-NH on 2 Kacem instances, 7 BRdata instances and 11 DPdata instances, and is significantly outperformed by MA-1-NH on only instances 15a and la35. As for the set coverage metric, MA-1 significantly outperforms MA-1-NH on 20 out of total 36 instances, and MA-1-NH is significant better than MA-1 also only on instances 15a and la35. The hierarchical strategy shows the similar power in MA-2. MA-2 performs significantly better than MA-2-NH on 24 out 36 instances in terms of both IGD metric and set coverage metric. MA-2-NH is significantly better than MA-2 only on the instance Mk05 in terms of IGD metric, and fails to significantly outperform MA-2 on any instance in terms of set coverage metric. Also of note is that the proposed hierarchical strategy seems to be especially beneficial for solving those instances with a larger number of non-dominated solutions, such as Mk09, Mk10, 11a and so on.

E. Comparison with State-of-the-Art Algorithms

In this subsection, the proposed MAs with excellent performance (MA-1 and MA-2) are compared with the existing state-of-the-art methods. To the best of our knowledge, there exists no algorithm for the MO-FJSP in the literature which is evaluated on all 36 problem instances considered in this paper. Thus, we choose those most representative ones as the benchmark algorithms on each data set. Moreover, as mentioned in Section I, in the literature on MO-FJSP, the performance of an algorithm is generally presented by listing all the non-dominated solutions collected over a certain number of runs. However, few algorithms report the results for each run. Hence, the statistical comparison just like that in the previous three subsections seems to be impossible to be conducted between the proposed MAs and the existing algorithms in the literature, although we think that this kind of comparison is more justified than the comparison of non-dominated solutions found over several runs. As regards the comparison in this part, for each instance and each algorithm, the metrics IGD and set coverage are computed for the set of non-dominated solutions collected over several runs. And the number of runs of MA-1 (MA-2) on each data set is set by the minimum number of runs used by the compared algorithms to avoid taking advantage of the compared algorithms.

In Tables VIII, IX, and X, MA-1 (MA-2) is compared on Kacem instances and BRdata instances with four algorithms recently proposed for the MO-FJSP, which are called HSFLA [23], PLS [24], SEA [29] and CMA [30] respectively in this paper. HSFLA is not evaluated on the instances ka4x5 and ka10x7, and PLS only considers 4 in 10 BRdata instances. Both HSFLA and PLS run 20 times for each instance, while SEA runs only 10 times. As for CMA, four variants of parameters are used, and each variant of CMA is run 10 times. So, CMA in fact runs total 40 times for each instance. The number of runs for the proposed MA-1 (MA-2) is set to 10.

First, it can be seen from Tables VIII, IX, and X that each of the six referred algorithms finds all the non-dominated solutions in the reference set for each Kacem instance over its defined number of runs. Next, we focuss only on BRdata instances. As for the IGD metric, MA-1 yields the best results on 7 out of 10 BRdata instances, and MA-2 achieves best on 8 out of 10 instances except Mk04 and Mk06. It seems that both of HSFLA and PLS perform relatively worse than the other four algorithms, because both of them are outperformed by the other algorithms on each BRdata instance in terms of IGD value. CMA obtains the best IGD value on the instance Mk06. But it should be noted that the results of CMA are based on many more number of runs. We also find that if the number of runs of MA-1 (MA-2) is extended to 30 times, the mentioned dominance of CMA on the instance Mk06 would disappear. MA-1 and MA-2 are compared respectively with the other algorithms in Tables IX and X using set coverage values. It is clearly indicated from the two tables that MA-1 (MA-2) is superior to all the compared algorithms in searching the non-dominated solutions. In Table XI, we compare the average running time consumed by MA-1, MA-2 and HSFLA. However, the different computing hardware, programming

TABLE VI
PERFORMANCE EVALUATION OF THE EFFECT OF HYBRIDIZING GENETIC SEARCH AND LOCAL SEARCH USING AVERAGE SET COVERAGE VALUES OVER 30 INDEPENDENT RUNS FOR ALL 36 PROBLEM INSTANCES.

Instance	MA-1 (A) vs MRLS-1 (C)		MA-1 (A) vs NSGA-II (E)		MA-2 (B) vs MRLS-2 (D)		MA-2 (B) vs NSGA-II (E)	
	$C(A,C)$	$C(C,A)$	$C(A,E)$	$C(E,A)$	$C(B,D)$	$C(D,B)$	$C(B,E)$	$C(E,B)$
ka4x5	**0.041667**	0.000000	0.000000	0.000000	0.016667	0.000000	0.000000	0.000000
ka08	**0.313333**	0.000000	0.008333	0.000000	**0.110000**	0.000000	0.008333	0.000000
ka10x7	0.000000	0.000000	0.033333	0.000000	0.000000	0.000000	0.033333	0.000000
ka10x10	0.033333	0.000000	**0.185000**	0.000000	0.008333	0.000000	**0.185000**	0.000000
ka15x10	**1.000000**	0.000000	**1.000000**	0.000000	**1.000000**	0.000000	**1.000000**	0.000000
Mk01	**0.943749**	0.000000	**0.747222**	0.000000	**0.945949**	0.000000	**0.752564**	0.016061
Mk02	**0.916005**	0.000000	**0.577838**	0.003704	**0.909114**	0.000000	**0.588899**	0.000000
Mk03	**1.000000**	0.000000	**0.025490**	0.000000	**1.000000**	0.000000	**0.025490**	0.000000
Mk04	**1.000000**	0.000000	**0.274072**	0.004000	**0.994444**	0.000000	**0.277258**	0.004615
Mk05	**0.896971**	0.000000	**0.070260**	0.003030	**0.780210**	0.000000	**0.076573**	0.000000
Mk06	**1.000000**	0.000000	**0.444094**	0.259674	**1.000000**	0.000000	**0.549810**	0.162620
Mk07	**1.000000**	0.000000	**0.176634**	0.000000	**1.000000**	0.000000	**0.176634**	0.006250
Mk08	**1.000000**	0.000000	0.000000	0.000000	**1.000000**	0.000000	0.000000	0.000000
Mk09	**1.000000**	0.000000	**0.808463**	0.025715	**1.000000**	0.000000	**0.824307**	0.012114
Mk10	**1.000000**	0.000000	**0.562177**	0.235564	**1.000000**	0.000000	**0.662532**	0.167871
01a	**1.000000**	0.000000	0.600000	0.316667	**1.000000**	0.000000	0.600000	0.400000
02a	**0.996296**	0.000000	**0.872778**	0.110370	**0.978782**	0.000000	**0.794444**	0.137077
03a	**0.994444**	0.000000	**0.937778**	0.011111	**0.974352**	0.010833	**0.891905**	0.064127
04a	**1.000000**	0.000000	**0.546229**	0.143587	**1.000000**	0.000000	**0.533477**	0.166952
05a	**1.000000**	0.000000	**0.754491**	0.134465	**1.000000**	0.000000	**0.790380**	0.098991
06a	**1.000000**	0.000000	**0.985997**	0.000641	**1.000000**	0.000000	**0.982198**	0.002925
07a	**1.000000**	0.000000	**0.616667**	0.311111	**1.000000**	0.000000	**0.764444**	0.173333
08a	**1.000000**	0.000000	**0.906111**	0.011111	**0.997436**	0.000000	**0.886111**	0.061706
09a	**1.000000**	0.000000	**0.843980**	0.018704	**1.000000**	0.000000	**0.973016**	0.000000
10a	**1.000000**	0.000000	**0.877519**	0.097341	**0.999660**	0.000000	**0.871207**	0.092518
11a	**1.000000**	0.000000	**0.813326**	0.066952	**1.000000**	0.000000	**0.856821**	0.041296
12a	**1.000000**	0.000000	**0.994903**	0.000290	**1.000000**	0.000000	**0.994900**	0.000233
13a	**1.000000**	0.000000	**0.529982**	0.170424	**1.000000**	0.000000	**0.447590**	0.215780
14a	**1.000000**	0.000000	**0.849749**	0.036574	**1.000000**	0.000000	**0.987963**	0.000000
15a	**1.000000**	0.000000	**0.805833**	0.047884	**1.000000**	0.000000	**0.993333**	0.000000
16a	**1.000000**	0.000000	**0.656102**	0.118672	**1.000000**	0.000000	**0.661786**	0.090860
17a	**1.000000**	0.000000	**0.945089**	0.005143	**1.000000**	0.000000	**0.966387**	0.000953
18a	**1.000000**	0.000000	**0.999274**	0.000000	**1.000000**	0.000000	**1.000000**	0.000000
la30	**1.000000**	0.000000	**0.891534**	0.010833	**1.000000**	0.000000	**0.947222**	0.014286
la35	**1.000000**	0.000000	**0.985000**	0.000000	**1.000000**	0.000000	**1.000000**	0.000000
la40	**1.000000**	0.000000	**0.823611**	0.008466	**1.000000**	0.000000	**0.936667**	0.000000

For each instance, the result that is significantly better than the other is marked in bold (with greater set coverage).

platforms and coding skills used in each algorithm make this comparison notoriously problematic [53]. Hence, we enclose the original name of the CPU, the programming language, and the original running time for the corresponding algorithm, which is enough for us to have a roughly understanding of the efficiency of referred algorithms. This practice has been often adopted in the existing research of the JSP [54], [55]. From Table XI, it seems that HSFLA is generally more computational expensive than the proposed MAs.

In Table XII, MA-1 (MA-2) is compared with MOGA [20] on DPdata instances. All the three algorithms are independently run 10 times. From the results of IGD, it can be observed that MA-1 (MA-2) outperforms MOGA on DPdata instances by a large margin. As for the computational time, MA-1 (MA-2) also shows superiority in most of considered instances. Moreover, he absolute dominance of MA-1 (MA-2) over MOGA can be more easily demonstrated in terms of set coverage values, which indicate that, for each instance, every solution obtained by MOGA is dominated by at least one solution by MA-1 (MA-2), and none of the solutions obtained by MA-1 (MA-2) is dominated by any solution by MOGA.

In Table XIII, the proposed MA-1 (MA-2) is compared with MOEA-GLS [17] on 3 Hurink Vdata instances. The number of runs for the three referred algorithms is set to 30. Because there exist only a few non-dominated solutions for the 3 instances, the solutions found over 30 independent runs by each algorithm are directly listed In Table XIII. For the instances la30 and la40, all the solutions obtained by MA-1 and MOEA-GLS are dominated by at least one solution from MA-2, and none of solutions obtained by MA-2 is dominated by solutions from the other two algorithms. For the instance la35, all the three algorithms find the same non-dominated solution. Table XIV indicates MA-1 (MA-2) seems much less time consuming than MOEA-GLS.

Although our MA-1 (MA-2) is specially proposed for the MO-FJSP, the minimization of makespan is considered mostly in them. Thus, in Table XV, we also compare their performance only in terms of the best makespan obtained with two state-of-the art algorithms for the SO-FJSP, which are TS of Mastrolilli and Gambardella [3], and CDDS of Hmida *et al.* [7]. The fourth column in Table XV lists the best known solution (BKS) ever reported in the literature for each instance. From Table XV, MA-1 (MA-2) achieves competitive performance with respect to TS and CDDS on

TABLE VII
PERFORMANCE EVALUATION OF THE EFFECT OF THE HIERARCHICAL STRATEGY IN LOCAL SEARCH USING AVERAGE IGD AND SET COVERAGE VALUES OVER 30 INDEPENDENT RUNS FOR ALL 36 PROBLEM INSTANCES.

Instance	IGD		MA-1 (A) vs MA-1-NH (C)		IGD		MA-2 (B) vs MA-2-NH (D)	
	MA-1	MA-1-NH	$C(A,C)$	$C(C,A)$	MA-2	MA-2-NH	$C(B,D)$	$C(D,B)$
ka4x5	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
ka08	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
ka10x7	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
ka10x10	**0.033793**	0.070134	**0.116667**	0.000000	0.092931	0.071523	**0.127778**	0.000000
ka15x10	**0.002778**	0.156178	**0.900000**	0.000000	**0.000000**	0.136906	**0.838889**	0.000000
Mk01	**0.001569**	0.022960	**0.489172**	0.000000	**0.001774**	0.015994	**0.348780**	0.016061
Mk02	**0.009685**	0.034411	**0.483089**	0.020370	**0.006263**	0.027675	**0.424478**	0.012500
Mk03	**0.000000**	0.000202	**0.025490**	0.000000	**0.000000**	0.000136	**0.019390**	0.000000
Mk04	**0.005937**	0.009063	**0.379759**	0.020682	**0.006307**	0.008421	**0.321268**	0.040832
Mk05	0.003265	0.000000	0.000000	0.003030	0.005062	**0.000000**	0.000000	0.003030
Mk06	**0.039592**	0.057145	**0.732628**	0.121220	**0.040446**	0.054947	**0.829041**	0.061756
Mk07	0.000000	0.001130	0.042361	0.000000	**0.000181**	0.002145	**0.065833**	0.000000
Mk08	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
Mk09	**0.002534**	0.017960	**0.908416**	0.006954	**0.001927**	0.019248	**0.921462**	0.005524
Mk10	**0.032299**	0.096543	**0.924833**	0.014070	**0.026703**	0.082840	**0.923299**	0.011510
01a	0.142513	0.130129	0.433333	0.533333	0.121680	0.114516	0.455556	0.500000
02a	0.029790	0.034487	0.509114	0.429167	**0.034413**	0.043247	0.559841	0.377315
03a	0.022955	0.027293	**0.606349**	0.263175	0.029485	0.031024	0.436905	0.471164
04a	**0.017067**	0.021816	0.389245	0.249457	0.020658	0.018123	0.275728	0.366822
05a	**0.029702**	0.044304	**0.811556**	0.118395	**0.027830**	0.042991	**0.877537**	0.077069
06a	**0.020006**	0.073221	**0.996160**	0.000000	**0.020205**	0.068212	**0.985961**	0.002807
07a	0.138625	0.137809	0.525000	0.392222	0.104688	0.116383	0.516667	0.405556
08a	**0.025773**	0.034817	0.538175	0.336746	0.038654	0.039831	0.495794	0.365913
09a	0.035028	0.036091	0.496890	0.425767	**0.026466**	0.038417	**0.613968**	0.272315
10a	**0.040129**	0.054284	**0.716120**	0.236064	**0.036169**	0.045667	**0.711205**	0.242898
11a	**0.024052**	0.109806	**0.998012**	0.000000	**0.020052**	0.111832	**0.997280**	0.000000
12a	**0.017390**	0.221964	**1.000000**	0.000000	**0.015395**	0.248510	**1.000000**	0.000000
13a	**0.058361**	0.073867	**0.621165**	0.148909	**0.061297**	0.091838	**0.707024**	0.213000
14a	0.041484	0.035127	0.352652	0.526402	**0.030463**	0.050004	**0.732103**	0.189431
15a	0.036633	**0.028674**	0.333018	**0.580291**	**0.021958**	0.028285	**0.606548**	0.295503
16a	**0.043449**	0.071278	**0.816502**	0.057209	**0.047622**	0.063839	**0.758392**	0.099110
17a	**0.022545**	0.184460	**1.000000**	0.000000	**0.015152**	0.214329	**1.000000**	0.000000
18a	**0.019398**	0.343793	**1.000000**	0.000000	**0.015005**	0.439499	**1.000000**	0.000000
la30	0.040453	0.042556	0.362513	0.494643	**0.035973**	0.046167	**0.717460**	0.203624
la35	0.016722	**0.012202**	0.266667	**0.627778**	0.012918	0.012686	0.405556	0.410000
la40	**0.028966**	0.042734	**0.684101**	0.250544	**0.016913**	0.059628	**0.953889**	0.020357

For each instance and each metric, the result that is significantly better than other is marked in bold (with smaller IGD, with greater set coverage).

TABLE VIII
COMPARISON BETWEEN THE PROPOSED MEMETIC ALGORITHMS AND THE EXISTING ALGORITHMS IN THE LITERATURE USING IGD VALUES OF THE SOLUTIONS FOUND OVER A CERTAIN NUMBER OF INDEPENDENT RUNS FOR KACEM INSTANCES AND BRDATA INSTANCES.

Instance	IGD					
	MA-1	MA-2	HSFLA	PLS	SEA	CMA
ka4x5	**0.000000**	**0.000000**	–	**0.000000**	**0.000000**	**0.000000**
ka08	**0.000000**	**0.000000**	**0.000000**	**0.000000**	**0.000000**	**0.000000**
ka10x7	**0.000000**	**0.000000**	–	**0.000000**	**0.000000**	**0.000000**
ka10x10	**0.000000**	**0.000000**	**0.000000**	**0.000000**	**0.000000**	**0.000000**
ka15x10	**0.000000**	**0.000000**	**0.000000**	**0.000000**	**0.000000**	**0.000000**
Mk01	**0.000000**	**0.000000**	0.202123	0.111817	0.010644	**0.000000**
Mk02	**0.000000**	**0.000000**	0.141176	0.060069	0.038961	0.009524
Mk03	**0.000000**	**0.000000**	0.222437	0.222437	**0.000000**	**0.000000**
Mk04	**0.002610**	0.002705	0.189767	–	0.012537	0.003653
Mk05	**0.000000**	**0.000000**	0.018541	–	0.024490	**0.000000**
Mk06	0.028061	0.025240	0.114675	–	0.053591	**0.022556**
Mk07	**0.000000**	**0.000000**	0.049865	–	0.015284	0.007078
Mk08	**0.000000**	**0.000000**	0.111717	0.111717	0.006982	**0.000000**
Mk09	0.000908	**0.000663**	0.118504	–	0.009631	0.001470
Mk10	0.019261	**0.015448**	0.086488	–	0.079933	0.044289

For each instance, the minimal IGD values obtained by the compared algorithms are marked in bold.

solving the SO-FJSP. A quite interesting note is that MA-1 (MA-2) even obtains new best known solutions to several instances. Specifically, MA-1 finds a new best known solution to the instance 05a, and MA-2 achieves 5 new best known solutions to the instances 06a, 11a, 12a, 17a and 18a. And for the instances 06a, 12a, 17a and 18a, the best known solutions are obviously improved by MA-2. The reason is yet to be found. It seems that the simultaneous optimization of multiple objectives in turn promotes the minimization of makespan for these instances.

From the above computational results and comparisons, it can be concluded that the proposed MA-1 and MA-2 generally outperform the existing state-of-the-art approaches for solving the MO-FJSP. Moreover, they also show the strong ability to minimize the makespan, and the best known makespan values for several problem instances are further improved.

VII. FURTHER DISCUSSIONS

In this section, we will give more in-depth explanation of the algorithm design combining some numerical results presented in Section VI. From Section VI-E, it can be seen

TABLE IX
COMPARISON BETWEEN MA-1 AND THE EXISTING ALGORITHMS IN THE LITERATURE USING SET COVERAGE VALUES OF THE SOLUTIONS FOUND OVER A CERTAIN NUMBER OF INDEPENDENT RUNS FOR KACEM INSTANCES AND BRDATA INSTANCES.

Instance	MA-1 (A) vs HSFLA (B)		MA-1 (A) vs PLS (C)		MA-1 (A) vs SEA (D)		MA-1 (A) vs CMA (E)	
	$C(A,B)$	$C(B,A)$	$C(A,C)$	$C(C,A)$	$C(A,D)$	$C(D,A)$	$C(A,E)$	$C(E,A)$
ka4x5	–	–	**0.000000**	**0.000000**	**0.000000**	**0.000000**	**0.000000**	**0.000000**
ka08	**0.000000**	**0.000000**	**0.000000**	**0.000000**	**0.000000**	**0.000000**	**0.000000**	**0.000000**
ka10x7	–	–	**0.000000**	**0.000000**	**0.000000**	**0.000000**	**0.000000**	**0.000000**
ka10x10	**0.000000**	**0.000000**	**0.000000**	**0.000000**	**0.000000**	**0.000000**	**0.000000**	**0.000000**
ka15x10	**0.000000**	**0.000000**	**0.000000**	**0.000000**	**0.000000**	**0.000000**	**0.000000**	**0.000000**
Mk01	**0.909091**	0.000000	**1.000000**	0.000000	**0.272727**	0.000000	**0.000000**	**0.000000**
Mk02	**1.000000**	0.000000	**0.750000**	0.000000	**0.142857**	0.000000	**0.250000**	0.000000
Mk03	**0.857143**	0.000000	**0.857143**	0.000000	**0.000000**	**0.000000**	**0.000000**	**0.000000**
Mk04	**1.000000**	0.000000	–	–	**0.100000**	0.000000	0.043478	**0.074074**
Mk05	**0.428571**	0.000000	–	–	**0.000000**	**0.000000**	**0.000000**	**0.000000**
Mk06	**1.000000**	0.000000	–	–	**0.669903**	0.009434	**0.598425**	0.179245
Mk07	**0.666667**	0.000000	–	–	**0.000000**	**0.000000**	**0.250000**	0.000000
Mk08	**0.625000**	0.000000	**0.625000**	0.000000	**0.000000**	**0.000000**	**0.000000**	**0.000000**
Mk09	**1.000000**	0.000000	–	–	**0.687500**	0.000000	**0.052632**	0.050000
Mk10	**0.733333**	0.035176	–	–	**0.934783**	0.010050	**0.762590**	0.120603

For each instance, the greater set coverage values obtained by the compared algorithms are marked in bold.

TABLE X
COMPARISON BETWEEN MA-2 AND THE EXISTING ALGORITHMS IN THE LITERATURE USING SET COVERAGE VALUES OF THE SOLUTIONS FOUND OVER A CERTAIN NUMBER OF INDEPENDENT RUNS FOR KACEM INSTANCES AND BRDATA INSTANCES.

Instance	MA-2 (A) vs HSFLA (B)		MA-2 (A) vs PLS (C)		MA-2 (A) vs SEA (D)		MA-2 (A) vs CMA (E)	
	$C(A,B)$	$C(B,A)$	$C(A,C)$	$C(C,A)$	$C(A,D)$	$C(D,A)$	$C(A,E)$	$C(E,A)$
ka4x5	–	–	**0.000000**	**0.000000**	**0.000000**	**0.000000**	**0.000000**	**0.000000**
ka08	**0.000000**	**0.000000**	**0.000000**	**0.000000**	**0.000000**	**0.000000**	**0.000000**	**0.000000**
ka10x7	–	–	**0.000000**	**0.000000**	**0.000000**	**0.000000**	**0.000000**	**0.000000**
ka10x10	**0.000000**	**0.000000**	**0.000000**	**0.000000**	**0.000000**	**0.000000**	**0.000000**	**0.000000**
ka15x10	**0.000000**	**0.000000**	**0.000000**	**0.000000**	**0.000000**	**0.000000**	**0.000000**	**0.000000**
Mk01	**0.909091**	0.000000	**1.000000**	0.000000	**0.272727**	0.000000	**0.000000**	**0.000000**
Mk02	**1.000000**	0.000000	**0.750000**	0.000000	**0.142857**	0.000000	**0.250000**	0.000000
Mk03	**0.857143**	0.000000	**0.857143**	0.000000	**0.000000**	**0.000000**	**0.000000**	**0.000000**
Mk04	**1.000000**	0.000000	–	–	**0.200000**	0.000000	**0.043478**	0.000000
Mk05	**0.428571**	0.000000	–	–	**0.000000**	**0.000000**	**0.000000**	**0.000000**
Mk06	**1.000000**	0.000000	–	–	**0.689320**	0.000000	**0.677165**	0.064220
Mk07	**0.666667**	0.000000	–	–	**0.000000**	**0.000000**	**0.250000**	0.000000
Mk08	**0.625000**	0.000000	**0.625000**	0.000000	**0.000000**	**0.000000**	**0.000000**	**0.000000**
Mk09	**1.000000**	0.000000	–	–	**0.703125**	0.000000	**0.052632**	0.016667
Mk10	**0.866667**	0.005181	–	–	**0.920290**	0.010363	**0.830935**	0.051813

For each instance, the greater set coverage values obtained by the compared algorithms are marked in bold.

that the proposed MAs can overcome the existing state-of-the-art algorithms. But why our MAs exhibit such excellent performance? Here, we would emphasise two special sides.

First, the neighbor generation in local search shows strong power in simultaneously decreasing the three objectives as much as possible. To be specific, the neighborhood search is driven by improving the makespan. We realize this by trying moving a critical operation in the current schedule one by one until an acceptable new schedule is obtained. The results in Table XV can partly indicate that the designed neighborhood structure is especially effective for minimizing the makespan, because our MAs are even competitive to the state-of-the-art algorithms for solving the SO-FJSP. However, it is not enough by considering only a single objective for the MO-FJSP. Since the makespan can be optimized by the property of critical operations, we further consider how to restrain the increasing of the other two objectives when decreasing the makespan. Just as described in Section V-B1, the neighbor of a schedule G is indeed generated by trying the actions in $\varphi(G)$. And once an action is executed successfully, the acceptable neighbor G' is formed. So, we sort the actions in $\varphi(G)$ according to the metrics Δ_t and Δ_c. Then, the actions are tried executing one by one according to the sorted order. The reason for doing this is that we want to make sure G' has the smallest possible total workload on the premise $C_{\max}(G') \leq C_{\max}(G)$. We give the least consideration to the objective critical workload, because only when actions have the same Δ_t values, Δ_c is further used to sort them. Then, why we deal with the three objectives according to this priority? Because the objective makespan is the hardest to be optimized among the three concerned objectives, which not only depends on the machine assignment for each operation but also depends on the operation sequence on each machine. Whether the makespan can be effectively minimized directly

TABLE XI
THE AVERAGE CPU TIME (IN SECONDS) CONSUMED BY MA-1, MA-2 AND HSFLA ON KACEM INSTANCES AND BRDATA INSTANCES.

Instance	MA-1[a]	MA-2[a]	HSFLA[b]
ka4x5	5.77	5.03	1.26
ka08	5.39	6.15	–
ka10x7	5.37	5.08	10.14
ka10x10	5.80	6.53	–
ka15x10	8.91	7.46	21.13
Mk01	20.30	20.16	172.18
Mk02	26.99	28.21	229.56
Mk03	56.60	53.76	139.87
Mk04	30.71	30.53	426.12
Mk05	37.50	36.36	153.12
Mk06	81.41	80.61	577.80
Mk07	38.54	37.74	185.23
Mk08	79.40	77.71	165.48
Mk09	74.74	75.23	565.70
Mk10	85.39	90.75	1072.20

[a] The CPU time on an Intel Core i7-3520M 2.9GHz processor in Java
[b] The CPU time on a Pentium IV 1.8GHz processor in C++

TABLE XII
COMPARISON BETWEEN THE PROPOSED MEMETIC ALGORITHMS AND MOGA USING IGD VALUES AND AVERAGE CPU TIME (IN SECONDS) FOR DPDATA INSTANCES.

Instance	MA-1[a]		MA-2[a]		MOGA[b]	
	IGD	CPU	IGD	CPU	IGD	CPU
01a	0.040359	198.33	**0.035874**	185.72	0.224215	122.50
02a	**0.004321**	155.34	0.016988	166.01	0.101119	153.40
03a	**0.007921**	150.80	0.013462	157.96	0.207378	174.00
04a	**0.006372**	121.34	0.011408	87.25	0.308259	124.20
05a	**0.014391**	121.04	0.014910	117.03	0.536145	142.40
06a	**0.012423**	138.70	0.013342	135.07	0.808465	185.60
07a	**0.022060**	200.52	0.039878	215.18	0.259504	457.80
08a	**0.003637**	122.56	0.010671	164.33	0.186362	496.00
09a	0.013067	106.50	**0.002006**	153.70	0.197739	609.60
10a	0.023802	188.23	**0.017439**	180.87	0.226811	452.80
11a	0.014375	176.87	**0.010544**	163.37	0.697549	608.20
12a	0.009872	192.45	**0.007111**	165.14	0.895897	715.40
13a	0.026774	195.87	**0.026426**	196.45	0.304963	1439.40
14a	0.024124	122.11	**0.009541**	153.67	0.352456	1743.20
15a	0.016359	123.93	**0.004840**	148.75	0.173609	1997.10
16a	0.026806	185.04	**0.023212**	194.71	0.394364	1291.40
17a	0.011310	184.15	**0.005050**	203.10	0.735482	1708.00
18a	0.008480	175.72	**0.006298**	191.23	0.922706	1980.40

[a] The CPU time on an Intel Core i7-3520M 2.9GHz processor in Java
[b] The CPU time on a 2GHz processor in C++
For each instance, the minimal IGD values obtained by the compared algorithms are marked in bold.

TABLE XIII
SOLUTIONS FOUND OVER 30 INDEPENDENT RUNS BY MA-1, MA-2 AND MOEA-GLS FOR 3 HURINK VDATA INSTANCES.

Instance	MA-1			MA-2			MOEA-GLS		
	$C_{\max}$	W_{T}	$W_{\max}$	$C_{\max}$	W_{T}	$W_{\max}$	$C_{\max}$	W_{T}	$W_{\max}$
la30	1076	10680	1075	**1072**	**10680**	**1072**	1075	10680	1075
	1078	10680	1074	**1073**	**10680**	**1071**	1077	10680	1073
	1079	10680	1073	**1082**	**10680**	**1070**	1079	10680	1072
	1080	10680	1072	**1135**	**10680**	**1069**			
	1086	10680	1071						
	1097	10680	1070						
la35	**1550**	**15485**	**1550**	**1550**	**15485**	**1550**	**1550**	**15485**	**1550**
la40	955	11472	772	**955**	**11472**	**769**	955	11472	783
	956	11472	771	**956**	**11472**	**768**	957	11472	780
	959	11472	770				963	11472	779
	967	11472	769				964	11472	777
							966	11472	775

For each instance, the solution that is not dominated by any other solution is marked in bold.

TABLE XIV
THE AVERAGE CPU TIME (IN SECONDS) CONSUMED BY MA-1, MA-2 AND MOEA-GLS ON 3 HURINK VDATA INSTANCES.

Instance	MA-1[a]	MA-2[a]	MOEA-GLS[b]
la30	47.95	53.43	2110.80
la35	67.29	83.03	355.20
la40	62.13	55.81	928.80

[a] The CPU time on an Intel Core i7-3520M 2.9GHz processor in Java
[b] The CPU time on a 2GHz processor in C++

influences the final algorithm performance. Moreover, it has been observed that the makespan and critical workload are approximately positively correlated on most of instances [30], so it may imply that minimizing the makespan is helpful to minimize the critical workload. This explains why we consider the total workload following the makespan. Our design enables that the makespan of schedules produced in the local search path is non-increasing, while the increasing of the other two objectives is suitably controlled, which makes a good comprise between the three objectives. In Section VI-C, the statistics show that MA-1 (MA-2) significantly outperforms the adapted NSGA-II, which verifies the effectiveness of the proposed local search. Further, in Section VI-D, we computationally show that considering three objectives hierarchically in the neighbor generation is better than just considering makespan.

Second, we would discuss the selection of the individuals from the population for local search. In order to reduce the computational effort for our MAs, we only apply local search to good individuals in the offspring population. And in each generation, only a predefined number of individuals are selected for local search, which is helpful to keep the diversity of the population. More specifically, the probability P_{ls} is defined to decide how many individuals are chosen. For choosing an individual, we use the tournament selection based on (5), where the weight vector is randomly drawn from a weight vector set. The ideas behind this design can be explained from the following two points of the view. One is to choose a good individual for a randomly specified local search direction instead of specifying a good local search direction to each individual [31]. The other is that we randomly choose the weight vector each time in order to promote the individuals close to the Pareto front in different directions by local search, which serves well the goals of convergence and diversity in the multiobjective optimization [56].

In summary, we implement a strong local search engine based on the neighbor generation procedure, and it is appropriately used to improve a portion of good individuals from the offspring population generated by genetic operators. The real effect is that the global-based genetic search provides good initial individuals for local search, and the improved individuals obtained by local search are well injected into the new population via non-dominated sorting and crowding distance, pulling the population permanently towards the Pareto

TABLE XV
COMPARISON OF THE BEST MAKESPAN OBTAINED ON ALL 36 PROBLEM INSTANCES.

Instance	BKS	MA-1	MA-2	TS	CDDS
ka4x5	11	**11**	**11**	–	–
ka08	14	**14**	**14**	–	–
ka10x7	11	**11**	**11**	–	–
ka10x10	**7**	**7**	–	–	
ka15x1	11	**11**	**11**	–	–
Mk01	40	**40**	**40**	**40**	**40**
Mk02	26	**26**	**26**	**26**	**26**
Mk03	204	**204**	**204**	**204**	**204**
Mk04	60	**60**	**60**	**60**	**60**
Mk05	172	**172**	**172**	173	173
Mk06	58	60	59	**58**	**58**
Mk07	139	**139**	**139**	144	**139**
Mk08	523	**523**	**523**	**523**	**523**
Mk09	307	**307**	**307**	**307**	**307**
Mk10	197	205	202	198	**197**
01a	2505	2520	2521	**2518**	**2518**
02a	2230	2236	2244	**2231**	**2231**
03a	2228	2231	2234	**2229**	**2229**
04a	2503	2510	2513	**2503**	**2503**
05a	2212	**2208**	2211	2216	2216
06a	2187	2173	**2172**	2203	2196
07a	2283	2371	2365	**2283**	**2283**
08a	2067	2083	2087	**2069**	**2069**
09a	2066	2081	2075	**2066**	**2066**
10a	2291	2340	2327	**2291**	**2291**
11a	2061	2067	**2057**	2063	2063
12a	2027	1998	**1992**	2034	2031
13a	2257	2306	2311	2260	**2257**
14a	2167	2192	2187	**2167**	**2167**
15a	2165	2186	2180	2167	**2165**
16a	2255	2292	2293	**2255**	2256
17a	2140	2129	**2119**	2141	2140
18a	2127	2086	**2077**	2137	2127
la30	1069	1076	1072	**1069**	–
la35	1549	1550	1550	**1549**	–
la40	955	**955**	**955**	**955**	–

For each instance, the minimal makespan values obtained by the compared algorithms are marked in bold.

front. This effect is computationally proved in Section VI-C, where the results indicate that the hybridization of genetic search and local search is significantly better than its individual component. It's worth noting that the two sides discussed above both concern the local search. But it doesn't mean that the genetic search is indifferent to our MAs. In fact, from the results shown in Section VI-C, MA-1 (MA-2) performs much better than MRLS-1 (MRLS-2), which demonstrates that the genetic search also plays a key role in the proposed MAs. However, just in the sense of algorithm design, it is not so delicate as well as local search.

VIII. CONCLUSION AND FUTURE WORK

In this paper, we study the multiobjective flexible job shop scheduling problem (MO-FJSP) with the makespan, total workload, and critical workload criteria, which has a strong industrial background and is very close to the real manufacturing situation. To propose effective memetic algorithms (MAs), we first adapt the classical NSGA-II to the MO-FJSP through well-designed chromosome encoding/decoding scheme and genetic operators. Then, a novel local search based on critical operations is specially developed for the MO-FJSP. It is worth noting that a hierarchical strategy is adopted in local search to enhance the ability to deal with multiple objectives. This strategy considers the minimization of makespan with the highest priority, while the total workload and critical workload are concerned subsequently by ordering all the possible actions that could generate the acceptable neighbor. Afterwards, the local search is embedded into the adapted NSGA-II to stress intensification, and the proposed MAs are formed. The two alternative acceptance rules in local search have a little influence on the performance of MAs, but the "Pareto" acceptance rule is preferred on the whole. To show how our MAs work, the effectiveness of key components in our MAs is also verified, including genetic search, local search, and the hierarchical strategy in local search. Moreover, extensive comparisons are carried out for the proposed MAs against the existing state-of-the-art algorithms on each data set. According to the computational results, the proposed MAs outperform all the other algorithms by a considerable margin. The success of our MA shows the effectiveness of combing traditional multiobjective evolutionary techniques with problem-specific search algorithms.

In the future, we will continue this study in the following directions. First, we would introduce a learning mechanism to dynamically adjust the local search probability at each generation in order to further improve the performance of our MAs. Second, we want to focuss on further exploring the problem-specific characteristics, and develop more effective local search for the MO-FJSP. Third, it could be interesting to adapt the proposed MAs to the other production scheduling models with multiple objectives [57]–[59].

ACKNOWLEDGMENT

The authors would like to thank the editors and reviewers for their valuable comments and suggestions. This work was supported by National Natural Science Foundation of China (Grant No. 61175110), National S&T Major Projects of China (Grant No. 2011ZX02101-004) and National Basic Research Program of China (973 Program) (Grant No. 2012CB316305).

REFERENCES

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

[2] P. Brandimarte, "Routing and scheduling in a flexible job shop by tabu search," *Annals of Operations Research*, vol. 41, no. 3, pp. 157–183, 1993.

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

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

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

[6] W. Bożejko, M. Uchroński, and M. Wodecki, "Parallel hybrid metaheuristics for the flexible job shop problem," *Computers & Industrial Engineering*, vol. 59, no. 2, pp. 323–333, 2010.

[7] A. Ben Hmida, M. Haouari, M.-J. Huguet, and P. Lopez, "Discrepancy search for the flexible job shop scheduling problem," *Computers & Operations Research*, vol. 37, no. 12, pp. 2192–2201, 2010.

[8] Y. Yuan, H. Xu, and J. Yang, "A hybrid harmony search algorithm for the flexible job shop scheduling problem," *Applied Soft Computing*, vol. 13, no. 7, pp. 3259–3272, 2013.

[9] Y. Yuan and H. Xu, "An integrated search heuristic for large-scale flexible job shop scheduling problems," *Computers & Operations Research*, vol. 40, no. 12, pp. 2864–2877, 2013.

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

[11] H. Liu, A. Abraham, O. Choi, and S. Moon, "Variable neighborhood particle swarm optimization for multi-objective flexible job-shop scheduling problems," *Simulated Evolution and Learning*, pp. 197–204, 2006.

[12] J. Gao, M. Gen, L. Sun, and X. Zhao, "A hybrid of genetic algorithm and bottleneck shifting for multiobjective flexible job shop scheduling problems," *Computers & Industrial Engineering*, vol. 53, no. 1, pp. 149–162, 2007.

[13] G. Zhang, X. Shao, P. Li, and L. Gao, "An effective hybrid particle swarm optimization algorithm for multi-objective flexible job-shop scheduling problem," *Computers & Industrial Engineering*, vol. 56, no. 4, pp. 1309–1318, 2009.

[14] L. Xing, Y. Chen, and K. Yang, "An efficient search method for multi-objective flexible job shop scheduling problems," *Journal of Intelligent Manufacturing*, vol. 20, no. 3, pp. 283–293, 2009.

[15] J. Li, Q. Pan, and Y. Liang, "An effective hybrid tabu search algorithm for multi-objective flexible job-shop scheduling problems," *Computers & Industrial Engineering*, vol. 59, no. 4, pp. 647–662, 2010.

[16] I. Kacem, S. Hammadi, and P. Borne, "Pareto-optimality approach for flexible job-shop scheduling problems: hybridization of evolutionary algorithms and fuzzy logic," *Mathematics and computers in simulation*, vol. 60, no. 3, pp. 245–276, 2002.

[17] N. Ho and J. Tay, "Solving multiple-objective flexible job shop problems by evolution and local search," *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, vol. 38, no. 5, pp. 674–685, 2008.

[18] M. Frutos, A. Olivera, and F. Tohmé, "A memetic algorithm based on a nsgaii scheme for the flexible job-shop scheduling problem," *Annals of Operations Research*, vol. 181, no. 1, pp. 745–765, 2010.

[19] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 2, pp. 182–197, 2002.

[20] X. Wang, L. Gao, C. Zhang, and X. Shao, "A multi-objective genetic algorithm based on immune and entropy principle for flexible job-shop scheduling problem," *International Journal of Advanced Manufacturing Technology*, vol. 51, no. 5, pp. 757–767, 2010.

[21] G. Moslehi and M. Mahnam, "A pareto approach to multi-objective flexible job-shop scheduling problem using particle swarm optimization and local search," *International Journal of Production Economics*, vol. 129, no. 1, pp. 14–22, 2011.

[22] J. Li, Q. Pan, and K. Gao, "Pareto-based discrete artificial bee colony algorithm for multi-objective flexible job shop scheduling problems," *International Journal of Advanced Manufacturing Technology*, vol. 55, no. 9, pp. 1159–1169, 2011.

[23] J. Li, Q. Pan, and S. Xie, "An effective shuffled frog-leaping algorithm for multi-objective flexible job shop scheduling problems," *Applied Mathematics and Computation*, vol. 218, no. 18, pp. 9353–9371, 2012.

[24] J. Li, Q. Pan, and J. Chen, "A hybrid pareto-based local search algorithm for multi-objective flexible job shop scheduling problems," *International Journal of Production Research*, vol. 50, no. 4, pp. 1063–1078, 2012.

[25] L. Wang, G. Zhou, Y. Xu, and M. Liu, "An enhanced pareto-based artificial bee colony algorithm for the multi-objective flexible job-shop scheduling," *International Journal of Advanced Manufacturing Technology*, vol. 60, no. 9, pp. 1111–1123, 2012.

[26] S. Rahmati, M. Zandieh, and M. Yazdani, "Developing two multi-objective evolutionary algorithms for the multi-objective flexible job shop scheduling problem," *International Journal of Advanced Manufacturing Technology*, In press.

[27] M. Rabiee, M. Zandieh, and P. Ramezani, "Bi-objective partial flexible job shop scheduling problem: NSGA-II, NRGA, MOGA and PAES approaches," *International Journal of Production Research*, vol. 50, no. 24, pp. 7327–7342, 2012.

[28] J. Xiong, X. Tan, K. Yang, L. Xing, and Y. Chen, "A hybrid multiobjective evolutionary approach for flexible job-shop scheduling problems," *Mathematical Problems in Engineering*, vol. 2012, 2012.

[29] T. Chiang and H. Lin, "A simple and effective evolutionary algorithm for multiobjective flexible job shop scheduling," *International Journal of Production Economics*, vol. 141, no. 1, pp. 87–98, 2013.

[30] T. Chiang and H. Lin, "Flexible job shop scheduling using a multiobjective memetic algorithm," *Advanced Intelligent Computing Theories and Applications. With Aspects of Artificial Intelligence*, pp. 49–56, 2012.

[31] H. Ishibuchi, T. Yoshida, and T. Murata, "Balance between genetic search and local search in memetic algorithms for multiobjective permutation flowshop scheduling," *IEEE Transactions on Evolutionary Computation*, vol. 7, no. 2, pp. 204–223, 2003.

[32] G. Minella, R. Ruiz, and M. Ciavotta, "A review and evaluation of multiobjective algorithms for the flowshop scheduling problem," *INFORMS Journal on Computing*, vol. 20, no. 3, pp. 451–471, 2008.

[33] Y. Mei, K. Tang, and X. Yao, "Decomposition-based memetic algorithm for multiobjective capacitated arc routing problem," *IEEE Transactions on Evolutionary Computation*, vol. 15, no. 2, pp. 151–165, 2011.

[34] L. Ke, Q. Zhang, and R. Battiti, "MOEA/D-ACO: A multiobjective evolutionary algorithm using decomposition and ant colony," *IEEE Transactions on Cybernetics*, in press.

[35] B. Roy and B. Sussmann, "Les problemes dordonnancement avec contraintes disjonctives," *Note ds*, vol. 9, 1964.

[36] M. Pinedo, *Scheduling: theory, algorithms, and systems*. Springer Science+ Business Media, 2012.

[37] N. Krasnogor and J. Smith, "A tutorial for competent memetic algorithms: model, taxonomy, and design issues," *IEEE Transactions on Evolutionary Computation*, vol. 9, no. 5, pp. 474–488, 2005.

[38] Y. Ong and A. Keane, "Meta-lamarckian learning in memetic algorithms," *IEEE Transactions on Evolutionary Computation*, vol. 8, no. 2, pp. 99–110, 2004.

[39] Y. Ong, M. Lim, N. Zhu, and K. Wong, "Classification of adaptive memetic algorithms: a comparative study," *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, vol. 36, no. 1, pp. 141–152, 2006.

[40] H. Ishibuchi and K. Narukawa, "Some issues on the implementation of local search in evolutionary multiobjective optimization," in *Genetic and Evolutionary Computation–GECCO 2004*. Springer, 2004, pp. 1246–1258.

[41] H. Ishibuchi and T. Murata, "A multi-objective genetic local search algorithm and its application to flowshop scheduling," *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, vol. 28, no. 3, pp. 392–403, 1998.

[42] K. Sindhya, K. Deb, and K. Miettinen, "Improving convergence of evolutionary multi-objective optimization with local search: a concurrent-hybrid algorithm," *Natural Computing*, vol. 10, no. 4, pp. 1407–1430, 2011.

[43] H. Ishibuchi, Y. Hitotsuyanagi, and Y. Nojima, "An empirical study on the specification of the local search application probability in multiobjective memetic algorithms," in *IEEE Congress on Evolutionary Computation, 2007. CEC 2007.*. IEEE, 2007, pp. 2788–2795.

[44] H. Ishibuchi, Y. Hitotsuyanagi, N. Tsukamoto, and Y. Nojima, "Use of heuristic local search for single-objective optimization in multiobjective memetic algorithms," *Parallel Problem Solving from Nature–PPSN X*, pp. 743–752, 2008.

[45] D. Garrett and D. Dasgupta, "An empirical comparison of memetic algorithm strategies on the multiobjective quadratic assignment problem," in *Computational intelligence in miulti-criteria decision-making, 2009. mcdm'09. ieee symposium on*. IEEE, 2009, pp. 80–87.

[46] R. Cheng, M. Gen, and Y. Tsujimura, "A tutorial survey of job-shop scheduling problems using genetic algorithmsłi. representation," *Computers & Industrial Engineering*, vol. 30, no. 4, pp. 983–997, 1996.

[47] I. Oliver, D. Smith, and J. Holland, "A study of permutation crossover operators on the traveling salesman problem," in *Proceedings of the Second International Conference on Genetic Algorithms on Genetic algorithms and their application*. L. Erlbaum Associates Inc., 1987, pp. 224–230.

[48] S. Dauzère-Pérès and J. Paulli, "An integrated approach for modeling and solving the general multiprocessor job-shop scheduling problem using tabu search," *Annals of Operations Research*, vol. 70, no. 0, pp. 281–306, Apr. 1997.

[49] J. Hurink, B. Jurisch, and M. Thole, "Tabu search for the job-shop scheduling problem with multi-purpose machines," *OR Spectrum*, vol. 15, no. 4, pp. 205–215, 1994.

[50] C. Coello and N. Cortés, "Solving multiobjective optimization problems using an artificial immune system," *Genetic Programming and Evolvable Machines*, vol. 6, no. 2, pp. 163–190, 2005.

[51] E. Zitzler, K. Deb, and L. Thiele, "Comparison of multiobjective evolutionary algorithms: Empirical results," *Evolutionary computation*, vol. 8, no. 2, pp. 173–195, 2000.

[52] J. Demšar, "Statistical comparisons of classifiers over multiple data sets," *The Journal of Machine Learning Research*, vol. 7, pp. 1–30, 2006.

[53] J. C. Beck, T. Feng, and J.-P. Watson, "Combining constraint programming and local search for job-shop scheduling," *INFORMS Journal on Computing*, vol. 23, no. 1, pp. 1–14, 2011.
[54] D. Sha and C.-Y. Hsu, "A hybrid particle swarm optimization for job shop scheduling problem," *Computers & Industrial Engineering*, vol. 51, no. 4, pp. 791–808, 2006.
[55] C. Y. Zhang, P. Li, Y. Rao, and Z. Guan, "A very fast TS/SA algorithm for the job shop scheduling problem," *Computers & Operations Research*, vol. 35, no. 1, pp. 282–294, 2008.
[56] M. Laumanns, L. Thiele, K. Deb, and E. Zitzler, "Combining convergence and diversity in evolutionary multiobjective optimization," *Evolutionary computation*, vol. 10, no. 3, pp. 263–282, 2002.
[57] B. Qian, L. Wang, D.-x. Huang, W.-l. Wang, and X. Wang, "An effective hybrid de-based algorithm for multi-objective flow shop scheduling with limited buffers," *Computers & Operations research*, vol. 36, no. 1, pp. 209–233, 2009.
[58] R. Qing-Dao-Er-Ji, Y. Wang, and X. Wang, "Inventory based two-objective job shop scheduling model and its hybrid genetic algorithm," *Applied Soft Computing*, vol. 13, no. 3, pp. 1400–1406, 2013.
[59] B. Alidaee and H. Li, "Parallel machine selection and job scheduling to minimize sum of machine holding cost, total machine time costs, and total tardiness costs," *IEEE Transactions on Automation Science and Engineering*, in press.

Yuan Yuan received his B.S. degree in software engineering from Southeast University, Nanjing, China, in 2010. He is currently pursuing the Ph.D. degree in the Department of Computer Science and Technology, Tsinghua University, Beijing, China.

His current research interests mainly include the evolutionary multiobjective optimization, intelligent scheduling and statistical machine learning techniques.

Hua Xu received his B.S. from Xi'an Jiaotong University in 1998. He received his M.S. and Ph.D. degrees from Tsinghua University in 2000 and 2003, respectively. Now he is an associate professor in the Department of Computer Science and Technology, Tsinghua University.

His research fields include the following aspects: Data Mining, Intelligent Information Processing and Advanced Process Controllers for IC manufacturing equipments. He has published over 50 academic papers, received 10 invention patents of advanced controller and is also the copyright owner of 6 software systems. He has achieved the 2nd Prize of National Science and Technology Progress of China, the 1st Prize of Beijing Science and Technology and the 3rd Prize of Chongqing Science and Technology.