Hindawi Publishing Corporation
Mathematical Problems in Engineering
Volume 2012, Article ID 478981, 27 pages
doi:10.1155/2012/478981

Research Article

A Hybrid Multiobjective Evolutionary Approach for Flexible Job-Shop Scheduling Problems

Jian Xiong,[1] Xu Tan,[2] Ke-wei Yang,[1] Li-ning Xing,[1] and Ying-wu Chen[1]

[1] *Department of Management, College of Information System and Management, National University of Defense Technology, Changsha, Hunan 410073, China*
[2] *School of Software, Shenzhen Institute of Information Technology, Shenzhen 518029, China*

Correspondence should be addressed to Jian Xiong, xiongjian1984@hotmail.com

Received 24 March 2012; Revised 24 May 2012; Accepted 25 May 2012

Academic Editor: Alex Elias-Zuniga

This paper addresses multiobjective flexible job-shop scheduling problem (FJSP) with three simultaneously considered objectives: minimizing makespan, minimizing total workload, and minimizing maximal workload. A hybrid multiobjective evolutionary approach (H-MOEA) is developed to solve the problem. According to the characteristic of FJSP, a modified crowding distance measure is introduced to maintain the diversity of individuals. In the proposed H-MOEA, well-designed chromosome representation and genetic operators are developed for FJSP. Moreover, a local search procedure based on critical path theory is incorporated in H-MOEA to improve the convergence ability of the algorithm. Experiment results on several well-known benchmark instances demonstrate the efficiency and stability of the proposed algorithm. The comparison with other recently published approaches validates that H-MOEA can obtain Pareto-optimal solutions with better quality and/or diversity.

1. Introduction

As its practical importance, job-shop scheduling problem (JSP) has received considerable attention in various application areas. JSP is one of the hardest combinatorial optimization problems and has been proven to be a NP-hard problem [1, 2]. Flexible job-shop scheduling problem (FJSP) is a generalization of the classic JSP, which allows one operation to be performed on more than one machine. Consequently, in addition to the operations scheduling, another issue that needs to be taken into account is the assignment problem, that is determining the assignment of operations to machines. Brucker and Schlie [3] were among the first to address this problem. According to the flexibility, FJSP can be generally categorized into total flexibility FJSP (T-FJSP) and partial flexibility FJSP (P-FJSP) [4, 5].

Due to the complexity of FJSP and its NP-hard characteristic, heuristic or metaheuristic methods are preferable to solve the problem. Integrated approaches have been often used in order to improve the performance of algorithms [6]. With simultaneous consideration of multiple objectives, in particular *makespan*, *total workload*, and *maximal workload*, the approaches can be generally categorized into three classes [7]: (1) Transform the multiobjective problem to a monoobjective problem by a weighted sum approach; (2) The non-Pareto approaches dealing with different objectives in a separated way; (3) The Pareto approaches based on the Pareto optimality concept.

In the first type of approaches mentioned above, objective function is linearly combined by weighed sum approach and is given as follows:

$$f = \omega_1 f_1 + \omega_2 f_2 + \omega_3 f_3, \tag{1.1}$$

where ω_1, ω_2, and ω_3, respectively, denote the weight coefficients for the three objective values, satisfying $\omega_1 + \omega_2 + \omega_3 = 1$. According to existing literature, most approaches can be categorized as this type. Xia and Wu [6] proposed a hybrid approach by combining particle swarm optimization algorithm with simulated annealing. Zhang et al. [8] also introduced a hybridized particle swarm optimization approach for multiobjective FJSP. Liu et al. [9] presented a multiswarm approach to multiobjective FJSP. Gao et al. [10, 11] proposed two hybrid genetic algorithms. Experiment results show that Gao's algorithms are efficient for multiobjective FJSP. Xing et al. [12] designed a simulation model to deal with multiobjective FJSP, and an efficient search method [13] was developed by the authors. Li et al. [14] proposed a hybrid tabu search algorithm integrating variable neighborhood search to solve multiobjective FJSP. Bagheri et al. [15] proposed an artificial immune algorithm in which a function of makespan was taken as the affinity evaluation. Vlcot and Billaut [16] proposed a tabu search algorithm for finding a set of nondominated solution, with a linear combination of objectives makespan and maximal lateness. Kacem et al. [4] proposed a controlled evolutionary approach for multiobjective FJSP. In the proposed algorithm, objectives of makespan and total workload were dealt with separately. In particular, makespan was taken as evaluation function when the generation index is even, otherwise, total workload was taken as evaluation function.

Compared with the approaches of transforming the problem to a single-objective one, the literature on Pareto-based approaches, especially the multiobjective evolutionary approaches, rather scant. Kacem et al. [5] proposed a Pareto approach based on the hybridization of fuzzy logic and evolutionary algorithms to solve the FJSP. Ho and Tay [17] presented an efficient approach combining evolutionary algorithm and guided local search. Frutos et al. [18] introduced a Memetic Algorithm based on the nondominated sorting genetic algorithm (NSGA-II) [19], in which a local search procedure (simulated annealing) was added. Wang et al. [20] solved FJSP by a multiobjective genetic algorithm based on immune and entropy principle. Li et al. [21] introduced a hybrid Pareto-based discrete artificial bee colony algorithm for solving multiobjective FJSP, and, in their subsequent research, Li et al. [22] presented a hybrid Pareto-based local search (PLS) embedding a variable neighborhood search-based self-adaptive strategy for the problem. Moslehi and Mahnam [23] proposed a Pareto approach hybridizing particle swarm and local search to solve multiobjective FJSP.

In this paper, we propose a hybrid multiobjective evolutionary algorithm (H-MOEA) to solve the FJSP. In the proposed H-MOEA, NSGA-II [19] is employed as the basic optimizer. In multiobjective optimization problems, convergency and diversity are two important

issues. The former indicates the algorithm's ability to find the true Pareto optimal solutions, and the latter reflects the algorithm's ability to find as much as possible different Pareto optimal solutions. However, in FJSP, there are not many Pareto optimal solutions in objective space. Thus, a modified crowding distance measure is developed to keep the diversity of the population. In order to improve the convergency of the algorithm, a local search based on critical path theory is incorporated into the H-MOEA. Additionally, well-designed chromosome representation and genetic operators are presented in the algorithm.

The remainder of this paper is organized as follows. In Section 2, we briefly introduce the problem formulation and optimization model of FJSP. Some preliminary concepts are presented in Section 3. The elements of the proposed H-MOEA are described in Section 4. The computational results and the comparison with other algorithms are reported in Section 5. Finally, some conclusion of our work is presented in the last section.

2. Problem Formulation

Similar to the classic JSP, FJSP takes into account the assignment of each operation to a machine and sets its starting and ending times. However, the task is more challenging than the classic one because it requires a proper selection of a machine from a set of machines to process each operation. A FJSP considering n jobs to be processed on m machines can be described as follows.

(1) There is a set of n jobs to be scheduled.

(2) The set of machines is denoted as $M = M_1, M_2, \ldots, M_m$, where m is the number of machines.

(3) Each job i consists of a sequence of n_i operations $O_{i1}, O_{i2}, \ldots, O_{in_i}$.

(4) The execution of operation O_{ij} requires one machine out of a set of given machines represented as $M_{ij} \in M$.

FJSP is to determine an assignment and a sequence of operations on machines to minimize

(1) makespan, which indicates the maximal completion time of all jobs, denoted as f_1;

(2) total workload, which represents the total working time of all machines, denoted as f_2;

(3) critical machine workload, which is the most workload among all machines, denoted as f_3;

Then, the multiobjective FJSP can be generally formulated as follows:

$$\begin{aligned} \text{obj.:} \quad & (1)\ \min f_1 = \text{makespan} \\ & (2)\ \min f_2 = \text{total workload} \\ & (3)\ \min f_3 = \max\ \text{workload} \\ \text{s.t.:} \quad & (1)\ \text{precedence constraint.} \end{aligned} \tag{2.1}$$

There are some assumptions and constraints in FJSP.

(1) Jobs are independent from each other.

(2) Machines are independent from each other.

(3) Setting up time of machines is negligible.

(4) Move time between operations is negligible.

(5) At a given time, a machine can execute at most one operation.

(6) No more than one operation of the same job can be executed at a time.

(7) There are no precedence constraints among the operations of different jobs.

3. Preliminary Concepts

3.1. Concept of Domination and Nondomination

Within the framework of multiobjective optimization, the concept of dominance is defined as follows: a solution S_1 dominates solution S_2 if S_1 is as good as S_2 on all objective measures and better than S_2 on at least one objective measure. Thus, in this paper, we formally define the schedule dominance as follows.

Definition 3.1. A schedule S_1 dominates schedule S_2 if $f_i(S_1) \leq f_i(S_2)$ $(i = 1, 2, 3)$, with strict inequality holding for at least one objective measure.

Thus, the nondominance concept can be defined as follows.

Definition 3.2. A schedule S^* is said to be a nondominated schedule if it is not dominated by any other individual in the population.

3.2. Concept of Critical Path

Feasible schedules of FJSP can be represented as a directed graph $G(O, A, E)$, where O is the operation set, represented as node set in the graph, A is the conjunctive arc set, and E is the disjunctive arc set. Thus, some graph theory is often integrated into the optimization algorithm of FJSP. In particular, local search procedure based on critical path theory is usually combined with heuristic or meta-heuristic.

Two important concepts in directed solution graph are earliest start time and latest start time of a node [11]. The first concept is used to indicate the earliest time at which an operation can be processed, while the second concept is related to the latest time at which the implementation of an operation can begin without delaying the makespan of a schedule. For an operation, if its earliest start time is equal to latest start time, any delay in the start of the operation will delay the completion of the schedule. Such an operation is critical to the completion of all the jobs on time [11]. A path from starting node to ending node in a graph that entirely consists of critical operations is called a critical path. The makespan of a schedule is equal to the length of the critical path [24].

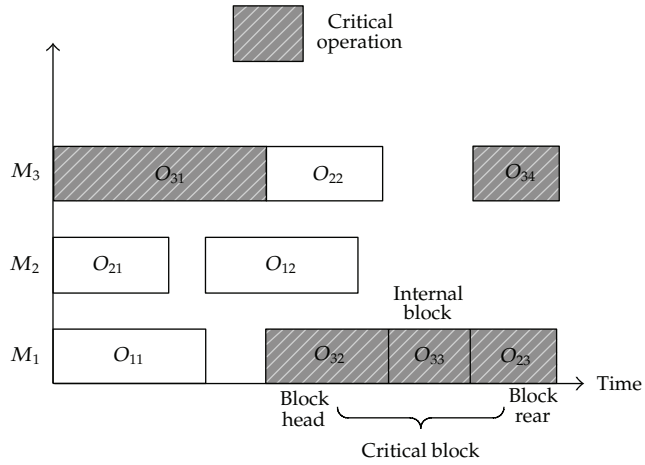

Figure 1: An example of critical block.

3.3. Concept of Critical Block

A *critical block* is a set of consecutive operations on the critical path processed by the same machine [17]. One machine can be associated with more than one critical block on a given critical path. The first operation of the critical block is called as *block head*, while the last operation is termed as *block rear*. The operation between block head and rear is called as *internal block*. Figure 1 shows an example of critical block.

4. Methodology

It is clear that a multiobjective approach is needed to solve the problem. To perform this task, one of the classic MOEAs, named NSGA-II [19], is employed as the multiobjective optimization algorithm. In the procedure of NSGA-II, the parent population and offsprings are combined and sorted in order to generate a population for the next generation. A nondominated sorting mechanism is performed to classify the combined population into different ranks of nondomination. A crowding-distance assignment is employed to ensure diversity is maintained among nondominated solutions. However, the crowding-distance mechanism is not very effective when there are few nondominated solutions and many-to-one mapping from decision space and objective space exists. Thus, a modified crowding-distance measure is introduced to maintain the diversity of the population. Additionally, chromosome representation, crossover, and mutation operators in original NSGA-II are redesigned for FJSP in this research.

4.1. Modified Crowding-Distance Measure

In multiobjective optimization problem, along with convergence to the Paretooptimal set, another issue that needs to be addressed is a good spread of solutions in the obtained set of solutions. To do this, the population should have a good diversity preservation. The original NSGA-II [19] employs a crowded-comparison approach to preserve the diversity

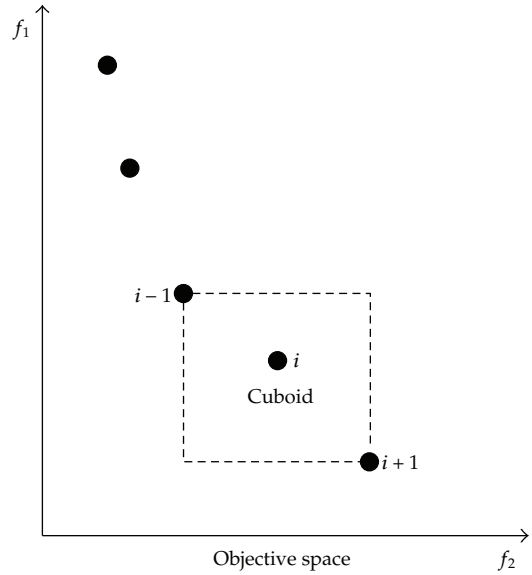

Figure 2: Crowding distance calculation.

of the population. The crowding distance of a solution is measured as the average distance of the solution from its nearest neighbors with the same nondomination rank in the objective space. Shown as in Figure 2, the crowding distance is calculated as the average side length of the cuboid.

However, in FJSP, there are not many nondominated solutions [17]. Furthermore, there exists many-to-one mapping from decision space to objective space. For example, two solutions with different machine assignment and/or operation sequence may have the same measure on objectives (an example will be shown in Section 5, Figures 7 and 8). In such a case, the crowding distance measure in original NSGA-II cannot effectively indicate the diversity estimation of an individual in the population. Shown as in Figure 3, suppose solutions A, B, and C in decision space have same machine assignments but different operation sequences, while solution D has different machine assignment and operation sequence from the other three solutions. In objective space, the four solutions have the same measures on all objectives. Based on the definition of nondomination solution, all of the four solutions will be maintained in the population. According to the crowding distance measure in the original NSGA-II, the four solutions have the same value on crowding distance. It indicates that the four solutions have the same priority to be selected to form the next population. However, it is reasonable to consider that solution D has a better contribution to the diversity of population. Thus, solution D should have a higher priority in the selection operator and the operator to form the population of next generation.

We modify the crowding distance as a measure in decision space. Such an idea has received much attention in recent years and been employed in some literatures related to multiobjective evolutionary algorithms [25–27]. Due to the stochastic feature of the nature-inspired algorithms, in FJSP, different solutions might have the same machine assignment. In order to keep the diversity of machine assignments, the crowding distance is measured by a function of the times of an assignment that occurs in the population. It is simply calculated

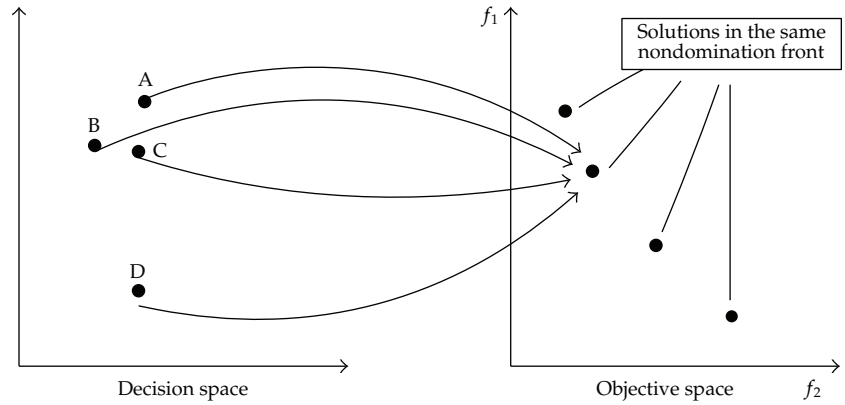

Figure 3: An example of many-to-one mapping from decision space to objective space.

as follows:

$$I[i]_{\text{distance}} = \frac{\text{popsize} - T[i]_{\text{assignment}}}{\text{popsize}}, \tag{4.1}$$

where popsize is the size of population, $T[i]_{\text{assignment}}$ represents the number of individuals in the population which have the same machine assignment with individual $I[i]$. According to the modified crowding distance measure, the solution with a diverse machine assignment will be selected first in the nondomination front to form the next population.

4.2. Chromosome Representation

Since FJSP has two independent tasks: machine assignment and operation scheduling, the chromosome representation of a solution should contain the information of these two parts. Permutation representation is often used to express operation sequence. In order to tackle the precedence constraints, Gen et al. [28] name all operations for a job with the same symbol and then interpret them according to the order of occurrences in the sequence of a given chromosome. This representation has been employed in FJSP, and a two-vector representation was proposed by Gao et al. [10, 11]. This paper also represents the chromosome in a two-vector approach, named machine assignment vector and operation sequence vector, but slightly modifies the representation of operation sequence.

In this study, the operation sequence is represented in a permutation way. Each operation is identified by a unique operation ID. A schedule is generated with the consideration of precedence constraints. Since we assumed that there are no precedence constraints among the operations of different jobs, each operation at most has one predecessor and successor, respectively. Figure 4 shows the modified two-vector chromosome representation of a FJSP with 4 jobs, 12 operations, and 4 machines. The length of each chromosome is equal to the total number of operations, denoted as O. For each position k in the chromosome, the machine assignment vector $v_1(k)$ represents the machine selected for the operation on position k, the operation sequence vector $v_2(k)$ denotes the ID index of the operation.

Machine assignment vector $v_1(k)$	M_1	M_3	M_3	M_2	M_1	M_4	M_2	M_3	M_2	M_4	M_1	M_2
Operation indicated	$O_{1,1}$	$O_{1,2}$	$O_{1,3}$	$O_{2,1}$	$O_{2,2}$	$O_{2,3}$	$O_{3,1}$	$O_{3,2}$	$O_{3,3}$	$O_{4,1}$	$O_{4,2}$	$O_{4,3}$
Operation sequence vector $v_2(k)$	1	2	3	4	5	6	7	8	9	10	11	12

Figure 4: An example of chromosome representation.

4.3. Population Initialization

The algorithm starts with initialization of population. For each individual, both machine assignment and operation sequence need to be initialized.

4.3.1. Machine Assignment Initialization

Machine assignment is initialized by a mix of different assignment rules. Here, we employ two assignment rules proposed in Kacem et al. [4].

(I) Assignment Rule 1: search for the global minimum in the processing time table.

(II) Assignment Rule 2: randomly permute jobs and machines in the processing time table.

Assignment Rule 1 starts from the operation that corresponds to the global minimum in the processing time table [29]. After several iterations, all assignments are found by entirely exploring the table of processing times [4]. Assignment Rule 2 randomly permutes the jobs and the machines before applying the approach by localization [29]. Thus, Assignment Rule 2 can avoid the inconvenience that tends to privilege some configurations as compared to the others [4]. The initial set of assignments is generated by making use of a mix of these two assignment rules. In the experiments of this research, we employ the same assignment initialization proportion as in [29], that is, 10% of initial population will be generated by Assignment Rule 1 and 90% of it will be generated by Assignment Rule 2. For details about these two rules, readers are referred to Kacem et al. [4] and Pezzella et al. [29].

4.3.2. Operation Sequence Initialization

Once the assignments are settled up, the sequence of operations on the machine needs to be determined. In the proposed H-MOEA, mix of the following four rules is used to select an operation from an eligible operation set.

(i) Long Processing Time (LPT), process the operation with maximal processing time firstly.

(ii) Most Work Remaining (MWR), process the operation among the task with the maximal total processing time firstly.

(iii) Most Operation Remaining (MOR), process the operation among the task with most remaining operation firstly.

(iv) Randomly Selection (RS), randomly select an operation to be scheduled from the eligible set.

4.4. Decoding

A shift is called global left-shift if some operations can be started earlier in time without delaying any other operations even though the shift has changed the operation sequence. A schedule is said to be active if no global left-shift exists [10]. Since makespan is taken as one of the optimization objective, in this study, we construct a schedule as active one, that is, the schedule is constructed as compact as possible.

In order to transfer the chromosome into an active schedule, several scheduling techniques are combined to generate the schedule. In concrete, we integrate Serial Sequence Generation Schedule (SSGS) [30, 31], scheduling algorithm proposed by Kacem et al. [4] and priority-based decoding method introduced by Gao et al. [11] in the decoding procedure. In the proposed decoding method, operations are scheduled at their earliest time from left to right according to their ID index in the chromosome. Algorithm 1 shows the proposed decoding procedure.

4.5. Crossover and Mutation

Crossover and mutation operators are used to generate the offspring, the former operator applies to pairs of chromosomes, while the latter operator applies to single one. Since the chromosome of FJSP consists of two vectors, genetic operators are executed to each part independently.

4.5.1. Assignment Operators

Assignment operators change the machine assignments of individuals, while the operation sequences are preserved.

We adopt crossover presented in Kacem et al. [4], shown as in Algorithm 2. N is the job number, and n_i and $n_{i'}$ are the operation numbers of job i and i', respectively.

It would be interesting to make genetic operators able to contribute in optimization objectives [4]. Along this avenue, we propose several mutation operators for machine assignment.

(1) Mutation operator balancing workload of machines, denoted as *Assignment_MO1*. This mutation operator is adopted from Kacem et al. [4] but made a slight modification. The operation on most loaded machine is randomly selected and assigned to another machine such that the workload of the new assigned machine cannot exceed the maximal workload of the individual before applying mutation operation.

(2) Mutation operator reducing total workload, denoted as *Assignment_MO2*. This mutation operator aims at contributing to optimization of total workload. A certain number of operations are randomly selected to be assigned on another machine (if exists), satisfying the following two conditions:

   (a) the processing time of the operation on the new assigned machine is less than the original machine;

   (b) the workload of the new assigned machine cannot exceed the maximal workload of the individual.

Beginning Decoding Procedure
initialize vector of machines availabilities *Dispo_Machine*[k]=0 for each machine M_k ($k \leq M$);
initialize vector of idle intervals *Interval_Machine*[k][m]=0 for each machine M_k ($k \leq M$);
initialize vector of jobs availabilities *Dispo_Job*[j]=0 for each job j($k \leq N$);

for i = 1 to O (O is the length of chromosome) **do**
  Schedule operations I_i ($1 \leq i \leq O$) from left to right.
  Search an available idle interval from left to right on machine k for operation I_i.
  if Such an interval exists. **then**
    Insert the operation into that interval.
    Update *Interval_Machine*[k][m].
  else
    Determine $t = \max\{Dispo\_Machine[k], Dispo\_Job[j]\}$.
    Set the start time of operation I_i as t.
    Update *Dispo_Machine*[k].
    Update *Interval_Machine*[k][m].
    Update *Interval_Machine*[k][m].
  end if
end for

Algorithm 1: Pseudocode of decoding procedure.

Crossover
Select randomly two parents P_1 and P_2;
Select randomly two integers i and i' such that $i \leq i' \leq N$;
Select randomly two integers j and j' such that $j \leq n_i$ and $j' \leq n_{i'}$ (in the case where $i = i'$, $j \leq j'$);
Child C_1 receives the same assignments from the parent P_1 for all operations between the operations O_{ij} and $O_{i'j'}$;
The rest of assignments for C_1 is obtained from P_2;
The construction of child C_2 is similar with C_1.

Algorithm 2: Procedure of crossover for assignment.

(3) Mutation operator reassigning critical operations, denoted as *Assignment_MO3*. In this operator, a certain number of critical operations are randomly selected and assigned to another machine on which the processing time cannot exceed the processing time on the original machine. In other words, the total workload after mutation is equal to or less than original one.

(4) Immigration mutation operator, denoted as *Assignment_MO4*. This operator is adopted from Gao et al. [11], which randomly generates new machine assignment for individuals according to the rules discussed in the stage of assignment initialization.

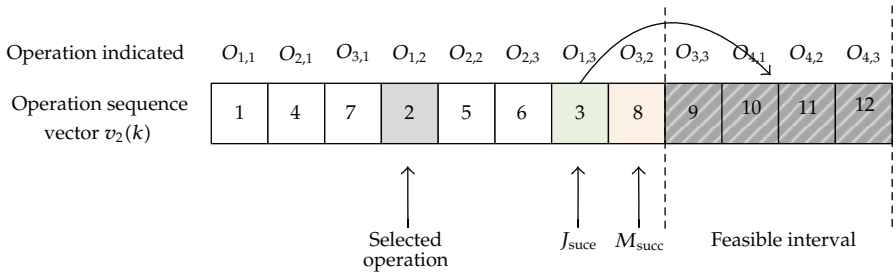

Figure 5: The first mutation operator for operation sequence.

4.5.2. Sequence Operators

In sequence operators, operation sequences are changed, while the machine assignments are preserved. Several crossover operators have been reported in the literature for permutation representation. In our algorithm, we apply a two-point position-based crossover [32, 33] for operation sequence. Usually, the two-point position-based crossover for scheduling problem can be divided into two versions [32]: in the first version, the activities outside the two selected points are inherited from one parent to the child, while in the other version, the set of activities between the two randomly selected points is inherited from one parent to the child. In this research, we employ the first version of crossover operator. In other words, for each crossover, two crossing points will be generated. We randomly generate these two crossing points from the interval $[1, O]$, denoted as r_1 and r_2, $r_1 \leq r_2$. Then, parent chromosome is separated as three parts by crossing points. For instance, we assume that two parent chromosomes P_1 and P_2 are selected for crossover, which can be represented as (P_1^1, P_1^2, P_1^3) and (P_2^1, P_2^2, P_2^3), respectively. Two children C_1 and C_2 are constructed by interchangeably inheriting different parts from parents, that is, C_1 takes the first and third parts from parent P_1, while inherits the second part from P_2. The process is the same for generating C_2, but with a reverse sequence $P_2^1 -> P_1^2 -> P_2^3$. What should be carefully noted is that, in order to preserve the precedence constraints, when inheriting the second and third parts, all operations that are already in the previous parts should be eliminated. This crossover operator is widely used in other scheduling problems, for example, project scheduling problem [34]. Hartmann [35] showed that this operator creates precedence feasible operation sequences.

Mutation operator generates new individuals slightly different from the parent. In this study, we adopt the procedure presented by Shadrokh and Kianfar [36] as the basic idea of mutation operator for operation sequence. In this mutation operator, an element in the operation sequence is randomly selected and inserted into another position between the interval determined by its last predecessor and first successor. Since this mutation operator is introduced for project scheduling with a permutation representation, it might not work effectively for FJSP in some cases. For example, when the machine predecessor and successor of the selected operation are not between its job predecessor and successor, changing the position of the selected operation will not impact the performance of an individual. Thus, we propose two mutation operators for FJSP based on the permutation representation.

The first mutation operator, denoted as *Sequence_MO1*, is used to permutate the position of an operation's job successor and/or job predecessor if its machine successor and/or machine predecessor are out of the interval of job predecessor and successor. Figure 5 shows an example of mutation operator *Sequence_MO1*. Suppose operation $O_{1,2}$ is the selected

operation to be applied mutation operator. The job successor and machine successor of $O_{1,2}$ are $O_{1,3}$ and $O_{3,2}$, with a index of 7 and 8, respectively, in the operation sequence. In such a case, permutating the position of $O_{1,2}$ in the sequence between its job predecessor and successor cannot change the schedule sequence of its machine successor $O_{3,2}$. Then, mutation operator *Sequence_MO1* is to permutate the position of the job successor $O_{1,3}$ to a feasible interval after the position of machine successor $O_{3,2}$ (if exists). The situation of predecessor is the same with successor.

The second mutation operator for operation sequence, denoted as *Sequence_MO*2, is defined as permutating the position of the first critical operation in a randomly selected interval from the operation sequence. This operator bears much similarity with the one proposed in Shadrokh and Kianfar [36]; the only difference is that *Sequence_MO*2 is only applied to critical operation.

4.6. Local Search Approach

The evolution speed of simple GAs is relatively slow [37]. One of the possible promising avenues to improve the convergence speed is the hybridization of local search [11]. Ishibuchi and Murata [38, 39] were among the first to implement such a hybridization. A central problem of any local search procedure for combinatorial optimization problem is how to define the effective neighborhood around an initial solution [10]. Since FJSP consists of two subproblems, that is, machine assignment and operation sequence, the neighborhood of a solution of FJSP should be defined to indicate the neighbors of these two parts.

4.6.1. Assignment Neighborhood

The makespan of a schedule is invariant when moving operations that are not on critical paths to other machines [17]. Thus, an efficient neighborhood of machine assignment is often constructed by adjusting critical operations on a machine to other available machines, such as in Li et al. [22]. In this study, an efficient assignment neighborhood is presented based on the idea of critical operation adjustment.

Step 1. Randomly select a machine M_s on which the workload is equal to the maximal workload of the individual.

Step 2. Select the critical operation(s) on machine M_s to be adjusted according to given probabilities based on the number of operations belonging to the same job. Suppose the vector of critical operations on machine M_s is denoted as $Operation\_M_s = (O'_{s1}, O'_{s2}, \ldots, O'_{sn'})$, the numbers of operations belonging to the same job with a critical operation on machine M_s are represented as $Num\_Operation\_M_s = (N_1, N_2, \ldots, N_{n'})$. Then, the probability vector of adjusting each critical operation is given as $Prob\_Adjust = (\alpha N_1, \alpha N_2, \ldots, \alpha N_{n'})$, where α is a coefficient, which is set as $1/OprNum\_Ave$, $OprNum\_Ave$ is the average number of operations for each job.

Step 3. For each critical operation on machine M_s selected to be executed for machine adjustment, assign the operation to a new machine M_{new} such that the combined value of total workload and maximal workload after adjustment is minimum. Let $T\_Workload_{\text{new}}$ and $M\_Workload_{\text{new}}$, respectively, denote the total workload and maximal workload of

Procedure: Sequence Adjustment
Randomly select a critical operation I_i.
if Operation I_i is critical block head **then**
  Randomly generate an integer i' between the machine successor and job successor of I_i.
  Insert operation I_i into the position i', $Insert(i, i')$.
end if
if Operation I_i is critical block rear **then**
  Randomly generate an integer i' between the job predecessor and machine predecessor of I_i.
  Insert operation I_i into the position i', $Insert(i, i')$.
end if
if Operation I_i is critical internal block **then**
  Randomly create a decimal d between 0 and 1.
  if $d \leq 0.5$ **then**
    Randomly generate an integer i' between the machine successor and job successor of I_i.
    Insert operation I_i into the position i', $Insert(i, i')$.
  else
    Randomly generate an integer i' between the job predecessor and machine predecessor of I_i.
    Insert operation I_i into the position i', $Insert(i, i')$.
  end if
end if

Algorithm 3: Procedure of sequence adjustment.

the individual after machine assignment. For a selected critical operation, search for a machine which is capable of processing the operation such that the combined function $f_{\text{combined}} = 0.1 \times T\_Workload_{\text{new}} + 0.9 \times M\_Workload_{\text{new}}$ is minimum among all possible machines.

4.6.2. Sequence Neighborhood

In order to define the neighborhood of operation sequence, critical path and/or critical block are often employed in FJSP by many researchers [8, 10, 11, 14, 17, 21, 22, 40]. Along this avenue, in this study, the effective neighborhood of operation sequence is based on the concept of critical block. A neighbor solution is obtained by adjusting the position of a randomly selected operation which belongs to a critical block. Algorithm 3 shows the procedure of sequence adjustment.

4.6.3. Framework of Local Search

Xing et al. [13] proposed an efficient search method for FJPS. In that proposed approach, two feasible moves were introduced to machine assignment, followed by a procedure of operation sequence. Then, a best move with a optimal performance can be obtained after feasible moves evaluation. Based on this approach, the present research proposes a local search procedure which incorporates machine adjustment and sequence adjustment. Algorithm 4

```
Procedure: Local search
for i = 0 to popsize do
    Ind_i: the ith individual in the population;
    rank_i: the value of non-domination rank of an individual Ind_i;
    JobNum: the number of job;
    TempInd: a new created temporary individual;
    Copy_Ind(Ind_i, TempInd);
    for j = 0 to JobNum/rank_i do
        Adjust machine assignment on Ind_i;
        Evaluate Ind_i;
        if Ind_i dominates TempInd then
            break;
        end if
        for k = 0 to JobNum/rank_i do
            Adjust operation sequence on Ind_i;
            Evaluate Ind_i;
            if Ind_i dominates TempInd then
                break;
            end if
        end for
        if Ind_i dominates TempInd then
            break;
        end if
        if TempInd dominates Ind_i then
            Copy_Ind(TempInd, Ind_i);
        end if
    end for
end for
```

Algorithm 4: Procedure of local search.

shows the procedure of the proposed local search. The local search procedure is implemented on each individual in the population. The size of neighborhood is set as dependent with the value of nondominated rank of an individual, given as follows:

$$N\,\text{Size}_i = \text{Round}\left(\frac{\text{JobNum}}{\text{rank}_i}\right), \tag{4.2}$$

where JobNum represents the number of jobs and rank_i indicates the nondominated rank value of the ith individual in the population, Round$(\cdot)$ is the function to obtain the integer part of a number. However, with the consideration of computational cost, this local search procedure is only implemented every dth generation. Several important issues in the hybridization of MOEAs and local search, such as balance between genetic search and local search [41, 42] and choice of solutions to which local search is applied [43], are left in our future work to study more efficient algorithms to solve larger scale problems.

4.7. Framework of H-MOEA

By hybridizing local search with multiobjective genetic algorithm, the convergence speed to local optimal can be improved. On the other hand, it might increase computational time per

iteration. Thus, a well-designed algorithm is very important to the real application cases. Figure 6 illustrates the framework of the proposed H-MOEA.

5. Computational Results

In this section, we compared the results obtained by the proposed hybrid multiobjective evolutionary approach (H-MOEA) and other recently published algorithms. In particular, we compared the results of AL+CGA and FL+EA by Kacem et al. [4, 5], PSO+SA by Xia and Wu [6], the hybrid genetic algorithm (hGA) by Gao et al. [11], the multiobjective evolutionary algorithm with local search (MOEA+GLS) by Ho and Tay [17], the multiobjective genetic algorithm (MOGA) proposed by Wang et al. [20], the PSO+TS by Zhang et al. [8], the artificial immune algorithm (AIA) by Bagheri et al. [15], the Pareto-based discrete artificial bee colony algorithm (P-DABC) by Li et al. [21], the Pareto-based local search algorithm (PLS) developed by Li et al. [22], and the multiobjective particle swarm optimization and local search method proposed by Moslehi and Mahnam [23]. We also compared the results obtained by the present approach with some algorithms developed by our group, such as local search method developed by Xing et al. [12].

5.1. Parameter Settings

The H-MOEA was implemented in C# on Core(TM)2 Duo CPU 2.66 GHz with Windows XP system. For each instances, 20 independent runs were performed. Depending on the complexity of the problems, the population size of the H-MOEA ranges from 40 to 400. The maximal number of generation is limited to 200. The parameter settings for other genetic algorithms are listed as follows.

(i) Assignment crossover probability: 75%.

(ii) Sequence crossover probability: 90%.

(iii) Assignment mutation *Assignment_MO1* probability: 45%.

(iv) Assignment mutation *Assignment_MO2* probability: 45%.

(v) Assignment mutation *Assignment_MO3* probability: 45%.

(vi) Assignment mutation *Assignment_MO4* probability: 10%.

(vii) Sequence mutation *Sequence_MO1* probability: 20%.

(viii) Sequence mutation *Sequence_MO2* probability: 20%.

Moreover, in the stage of operation sequence initialization, the percentages of different rules were set as follows: Long Processing Time (LPT) 30%; Most Work Remaining (MWR) 30%; Most Operation Remaining (MOR) 30%, Randomly Selection (RS) 10%. The local search procedure is applied every 10th generation.

5.2. Test Instances I

The first five test instances, range from 4 jobs × 5 machines to 15 jobs × 10 machines, were taken from Kacem. Release date was not considered in this set of test instances. The three simultaneously considered objectives are, respectively, denoted as f_1 (minimization of

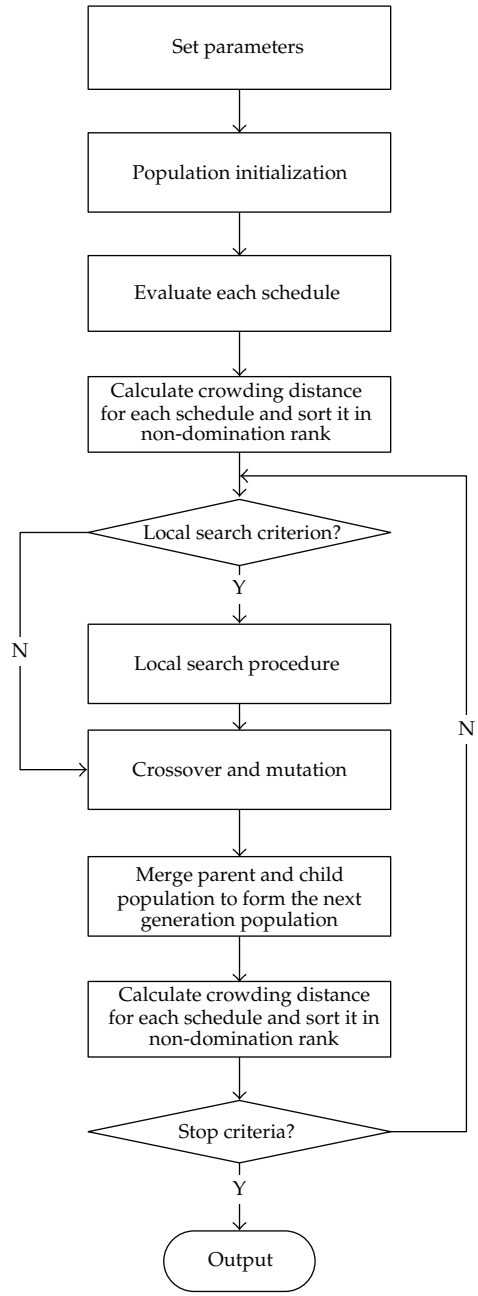

Figure 6: The flowchart of the proposed H-MOEA.

makespan), f_2 (minimization of total workload), and f_3 (minimization of critical workload). Table 1 shows the comparison of the obtained results on the five instances. In order to investigate the performance of proposed K-MOGA, we preprint the solutions dominated by the ones obtained by K-MOEA in italic font. From Table 1, one can see that our algorithm

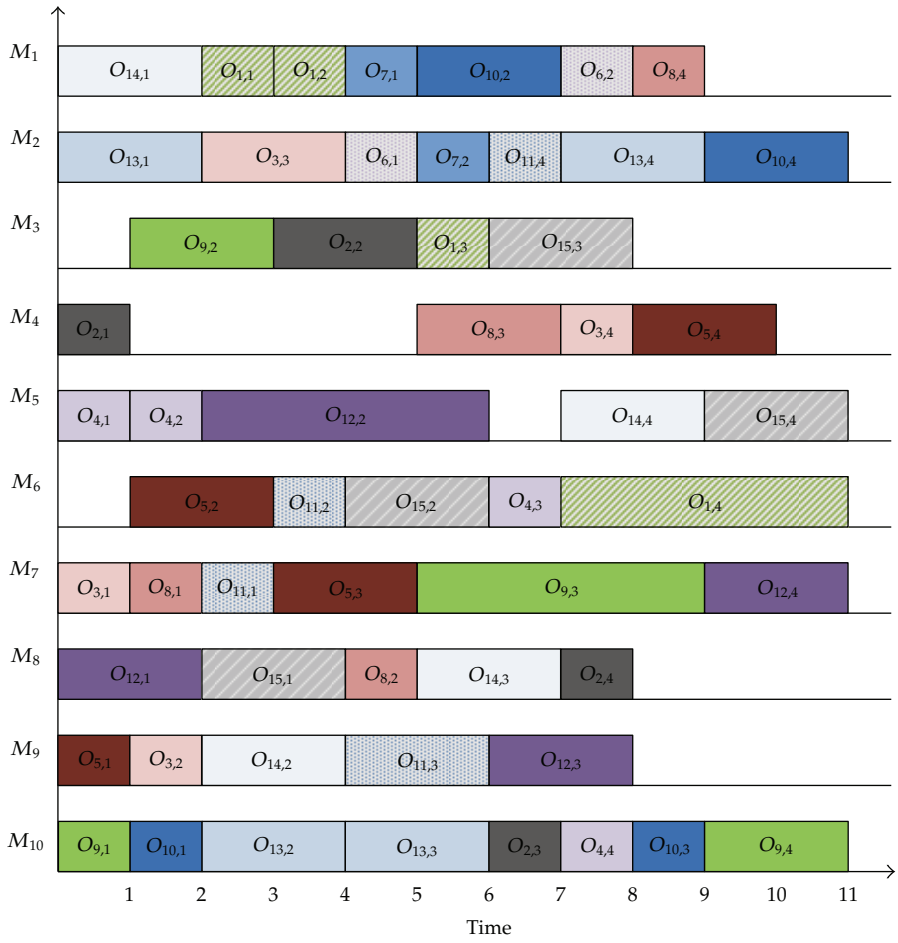

Figure 7: A solution of problem 15 × 10 (makespan = 11, total workload = 91, maximal workload = 11).

performs better for at least one problem instance, compared to all other algorithms expect MOEA-GLS. By looking at the results obtained by MOEA-GLS and our algorithm, it is shown that our algorithm is competitive to MOEA-GLS, except in problem 8 × 8 where 4 nondominated solutions were obtained by MOEA-GLS, while just 3 ones were obtained by our algorithm. The average processing time (in second), denoted as T, with 20 runs is reported in the last column in Table 1. The values of T indicate the efficiency of our algorithm.

As discussed earlier, it exits many-to-one mapping from decision space to objective space in FJSP. Figures 7 and 8 show two solutions of 15 × 10 instance. These two solutions have the same objective values, makespan 11, total workload 91, and maximal workload 11, but with different machine assignments. This indicates that keeping diversity in decision space could contribute to the convergency of the algorithm towards to Paretooptimal front.

The performance analysis of the algorithms not only considers the end state, but it is quite important to understand how they behave during the time of evolution. In this paper, we propose to measure the mean ideal distance (MID) [44] over time. MID is used to evaluate the closeness of solutions in a nondominated set with an ideal point which is

Table 1: Comparison of the Kacem instances without release date.

Size	f	PSO + SA		PSO + TS		P-DABC			MOPSO + LS					MOEA-GLS				MOGA				H-MOEA				T (s)
		1	2	1	2	1	2	3	1	2	3	4	5	1	2	3	4	1	2	3	4	1	2	3	4	
4×5	f_1			11		11	12	13										11	11	12		11	12	13	11	
	f_2			32		32	32	33										32	34	32		32	32	33	34	3.09
	f_3			10		10	8	7										10	9	8		10	8	7	9	
8×8	f_1	15	16	14	15	14	15	16	15	16	14	*16*	*17*	16	15	14	16	15	15	16		16	15	16		
	f_2	75	73	*77*	75	*77*	75	73	75	73	*77*	*78*	*77*	73	75	*77*	*77*	81	75	73		73	75	*77*		9.69
	f_3	12	13	12	12	12	12	13	12	13	12	*11*	*11*	13	12	12	11	11	12	13		13	12	11		
10×7	f_1					*12*	*11*	12														12	11	11		
	f_2					*61*	*63*	60														60	61	62		14.52
	f_3					*11*	*11*	12														12	11	10		
10×10	f_1	7		*7*		8	7	8	8	*7*	8	7		8	7	8	7	8	7	8	7	8	7	8	7	
	f_2	44		43		41	43	42	41	42	42	43		41	42	42	43	42	42	41	*45*	41	42	42	43	14.41
	f_3	6		6		7	5	5	7	6	5	5		7	6	5	5	5	6	7	*5*	7	6	5	5	
15×10	f_1	*12*		*11*		*12*	*11*		*12*	11				11	11			11	*12*	*11*		11	11			
	f_2	*91*		*93*		*91*	*93*		*93*	91				93	91			91	*95*	*98*		93	91			32.46
	f_3	*11*		*11*		*11*	*11*		*10*	11				10	11			11	*10*	*10*		10	11			

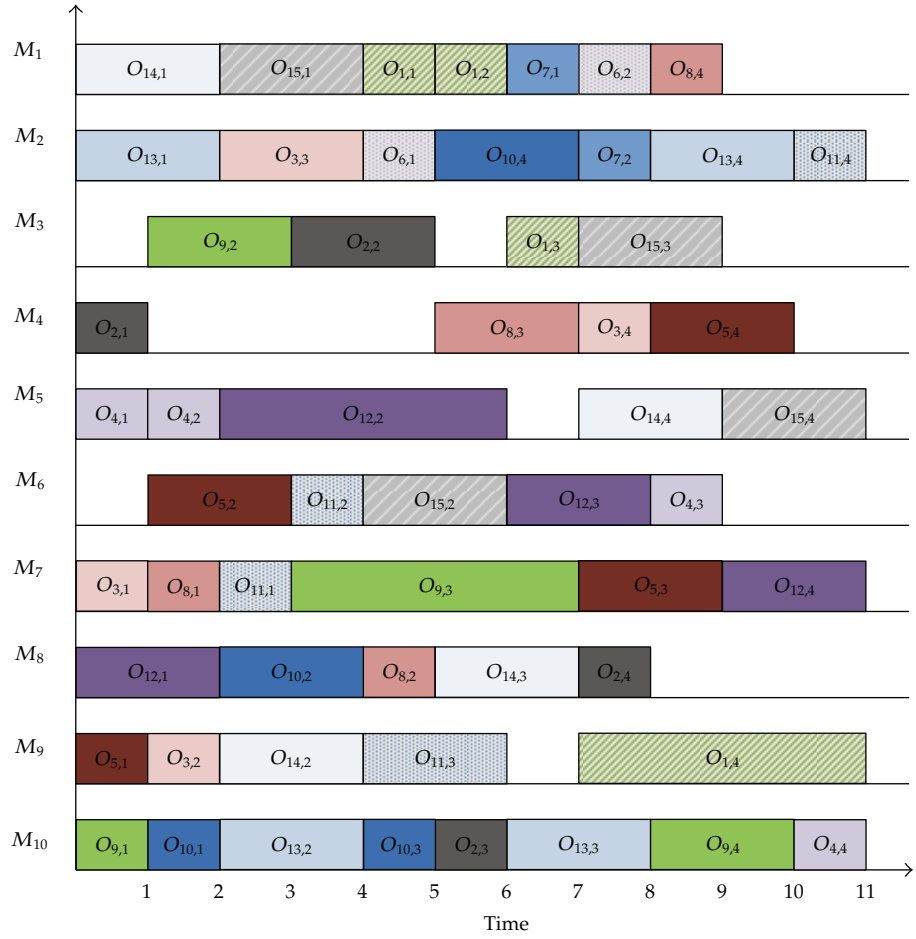

Figure 8: A solution of problem 15 × 10 with different machine assignment and operation sequence (makespan = 11, total workload = 91, maximal workload = 11).

usually considered as (0,0,0) [45]. MID can be calculated as follows:

$$\text{MID} = \sum_{i}^{\text{NoN}} d_i, \tag{5.1}$$

where NoN is the number of nondominated solutions, and d_i represents the distance of the ith nondominated solution from the ideal point.

Figure 9 depicts the performance curve of MID for instance 15 × 10 without release dates. The X-axis indicates the evolving generation, and the Y-axis represents the average value of MID over 20 independent runs. To such an instance, H-MOEA converged quickly in the first 20 generation. The first nondominated solution included in the final population was found at 32 generation, and all of the final nondominated solutions were identified at 46 generation. This suggests that the proposed H-MOEA is very efficient in solving FJSPs.

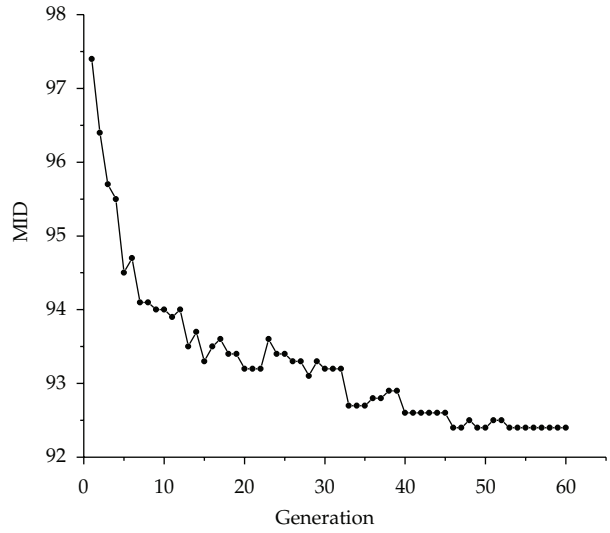

Figure 9: The performance curve of MID for 15 × 10 size test instance without release dates.

Table 2: Comparison of the Kacem instances with release date.

Size	f	FL + EA					P-DABC			MOPSO + LS			MOEA-GLS			H-MOEA			T (s)
		1	2	3	4	5	1	2	3	1	2	3	1	2	3	1	2	3	
4 × 5	f_1	18	18	16	16		16	16		16	16		16	16		16	16		
	f_2	32	33	35	34		32	33		32	33		32	33		32	33		0.57
	f_3	8	7	9	10		8	7		8	7		8	7		8	7		
8 × 8	f_1						20	20	20							20	20	20	
	f_2						73	75	77							73	75	77	9.93
	f_3						13	12	11							13	12	11	
10 × 7	f_1	16	15	18	17	16	15	16	15	16	15	15	15	16	15	15	16	15	
	f_2	60	61	63	64	66	61	60	62	60	61	62	61	60	62	61	60	62	13.89
	f_3	12	11	10	10	10	11	12	10	12	11	10	11	12	10	11	12	10	
10 × 10	f_1						12	13	13							13	13		
	f_2						43	41	42							41	42		10.23
	f_3						6	7	5							7	5		
15 × 10	f_1	24	23				23	23		23			23	23		23	23		
	f_2	91	95				91	93		91			91	93		91	93		15.88
	f_3	11	11				11	10		11			11	10		11	10		

5.3. Test Instances II

We also tested our algorithm on the five instances taken from Kacem et al. [4, 5], with consideration of release dates. The considered release date constraints are given as follows.

(1) Instance 4 × 5: $r_1 = 3$, $r_2 = 5$, $r_3 = 1$, $r_4 = 6$.

(2) Instance 8 × 8: $r_1 = 2$, $r_2 = 5$, $r_3 = 8$, $r_4 = 3$, $r_5 = 1$, $r_6 = 5$, $r_7 = 7$, $r_8 = 0$.

(3) Instance 10 × 7: $r_1 = 2$, $r_2 = 4$, $r_3 = 9$, $r_4 = 6$, $r_5 = 7$, $r_6 = 5$, $r_7 = 7$, $r_8 = 4$, $r_9 = 1$, $r_{10} = 0$.

Table 3: Comparison of the MK01 instance.

	Solution	f_1	f_2	f_3	T (s)
X-SM	1	*42*	*162*	*42*	286.80
AIA	1	*40*	*171*	*36*	97.21
PLS	1	40	167	36	N/A
	2	41	160	39	
	3	*42*	*163*	*37*	
	4	42	157	40	
	5	43	155	40	
	6	*45*	*154*	*40*	
	7	46	153	42	
HSFLA	1	40	167	36	172.18
	2	40	165	37	
	3	41	164	40	
	4	42	163	42	
	5	*42*	*164*	*37*	
	6	*43*	*162*	*38*	
	7	*44*	*166*	*36*	
	8	*44*	*160*	*38*	
	9	*46*	*163*	*37*	
	10	*48*	*165*	*36*	
	11	70	153	70	
H-MOEA	1	40	167	36	47.27
	2	40	165	37	
	3	41	161	38	
	4	**41**	**163**	**37**	
	5	41	168	36	
	6	**42**	**160**	**38**	
	7	**42**	**165**	**36**	
	8	**41**	**163**	**37**	
	9	42	157	40	
	10	42	158	39	
	11	43	155	40	
	12	**44**	**154**	**40**	
	13	46	153	42	

(4) Instance 10×10: $r_1 = 2$, $r_2 = 4$, $r_3 = 9$, $r_4 = 6$, $r_5 = 7$, $r_6 = 5$, $r_7 = 7$, $r_8 = 4$, $r_9 = 1$, $r_{10} = 0$.

(5) Instance 15×10: $r_1 = 5$, $r_2 = 3$, $r_3 = 6$, $r_4 = 4$, $r_5 = 9$, $r_6 = 7$, $r_7 = 1$, $r_8 = 2$, $r_9 = 9$, $r_{10} = 0$, $r_{11} = 14$, $r_{12} = 13$, $r_{13} = 11$, $r_{14} = 12$, $r_{15} = 5$.

Table 2 gives the comparison results with several recently published algorithms. It can be seen from Table 2 that our algorithm can obtain better or equiponderant solutions compared with others, except for instance 10×10. In the problem 10×10 with consideration of release date, our algorithms obtained two nondominated solutions with an average time 10.21 s. The algorithm P-DABC proposed by Li et al. [21] could obtain 3 nondominated solutions. However, since the authors of P-DABC did not report the average processing time

Table 4: Comparison of the MK02 instance.

	Solution	f_1	f_2	f_3	T(s)
X-SM	1	*28*	*155*	*28*	181.2
AIA	1	*26*	*154*	*26*	103.46
PLS	1	**26**	**151**	**26**	N/A
	2	*27*	*146*	*27*	
	3	*28*	*145*	*28*	
	4	*29*	*145*	*27*	
	5	*29*	*144*	*28*	
	6	29	143	29	
	7	*31*	*142*	*30*	
	8	33	140	33	
HSFLA	1	26	152	26	229.56
	2	*27*	*151*	*26*	
	3	*27*	*150*	*27*	
	4	*28*	*146*	*28*	
	5	*28*	*148*	*27*	
	6	*29*	*147*	*27*	
	7	*29*	*145*	*28*	
	8	*30*	*146*	*27*	
	9	*30*	*144*	*30*	
	10	*33*	*150*	*26*	
	11	*33*	*141*	*33*	
	12	*34*	*141*	*31*	
	13	*36*	*140*	*36*	
	14	*36*	*142*	*30*	
	15	*38*	*140*	*33*	
H-MOEA	1	*26*	*152*	*26*	51.26
	2	27	150	26	
	3	**27**	**145**	**27**	
	4	**28**	**144**	**28**	
	5	**29**	**143**	**29**	
	6	**30**	**142**	**30**	
	7	**31**	**141**	**31**	
	8	**33**	**140**	**33**	

on this instance, it could not be considered that our algorithm performed worse on obtaining diverse nondominated solutions.

5.4. Test Instances III

In order to make a further investigation on the performance of our algorithm, we ran H-MOEA on several instances from the BR data set presented by Brandimarte [46]. We used the same four different scale instances as in Li et al. [21] to verify the performance of H-MOEA. Four recently published algorithms were compared to our algorithm. The first algorithm was developed by the researchers in our group Xing et al. [13], denoted as X-SM. The second algorithm AIA was proposed by Bagheri et al. [15]. The third one, which was also the most

Table 5: Comparison of the MK03 instance.

	Solution	f_1	f_2	f_3	T(s)
X-SM	1	*204*	*852*	*204*	1568.40
AIA	1	*204*	*1207*	*204*	247.37
PLS	1	*204*	*852*	*204*	N/A
	2	*210*	*850*	*204*	
	3	*213*	*846*	*213*	
	4	*220*	*848*	*210*	
	5	*222*	*842*	*222*	
	6	*223*	*844*	*213*	
	7	330	812	330	
HSFLA	1	*204*	*852*	*204*	139.87
	2	*210*	*850*	*204*	
	3	*213*	*846*	*213*	
	4	*220*	*848*	*210*	
	5	*222*	*842*	*222*	
	6	*223*	*844*	*213*	
	7	330	812	330	
H-MOEA	1	**204**	**850**	**204**	318.13
	2	**210**	**848**	**210**	
	3	**213**	**844**	**213**	
	4	**221**	**842**	**221**	
	5	**222**	**838**	**222**	
	6	231	834	231	
	7	240	832	240	
	8	258	828	258	
	9	267	826	267	
	10	303	818	303	
	11	312	816	312	
	12	321	814	321	
	13	330	812	330	

recently published one, was introduced by Li et al. [22] and denoted as PLS. The fourth algorithm, which was also the most recently published one, hybrid shuffled frog-leaping algorithm (HSFLA), was proposed by Li et al. [47]. The comparison results are reported in Tables 3, 4, 5, and 6. In order to show the performance of our algorithm in a more clear way, we also preprint the solutions dominated by other ones in *italic font* and preprint the solutions dominating at least one other solution in **bold face**.

As shown in Tables 3 to 6, compared with X-SM, H-MOEA could obtain dominating or nondominated solutions with less computational effort. The solutions obtained by AIA for all tested instances were dominated by the solutions by our algorithm. By looking at the Pareto-based approach PLS, one can see that most of solutions obtained by PLS were dominated by the ones obtained by our algorithm, while in reverse, just one solution obtained by our algorithm for instance MK02 was dominated by the solution obtained by PLS. Also, it can be seen that H-MOEA could obtain a richer nondominated set than PLS. Since the computational time of PLS was not reported by the authors, further comparison and more accurate conclusion about the performances of the two algorithms could not be

Table 6: Comparison of the MK08 instance.

	Solution	f_1	f_2	f_3	T (s)
X-SM	1	523	2524	523	4060.20
AIA	1	*523*	*2723*	*523*	392.25
PLS	1	523	2524	523	N/A
	2	524	2519	524	
	2	533	2514	533	
	3	*548*	*2509*	*542*	
	4	*576*	*2507*	*569*	
	5	*578*	*2502*	*578*	
	6	*587*	*2497*	*587*	
	7	*594*	*2484*	*587*	
HSFLA	1	523	2524	523	165.48
	2	524	2519	524	
	3	533	2514	533	
	4	*548*	*2509*	*548*	
	5	*576*	*2507*	*569*	
	6	*578*	*2502*	*578*	
	7	*587*	*2497*	*587*	
	8	*594*	*2484*	*587*	
H-MOEA	1	523	2524	523	1674.14
	2	524	2519	524	
	3	533	2514	533	
	4	**542**	**2509**	**542**	
	5	**551**	**2504**	**551**	
	6	**560**	**2499**	**560**	
	7	**569**	**2494**	**569**	
	8	**578**	**2489**	**578**	
	9	**587**	**2484**	**587**	

available. The comparison between HSFLA and the proposed H-MOEA shows that most of nondominated solutions obtained by HSFLA were dominated by the solutions obtained by H-MOEA. It suggests that H-MOEA has better convergence ability than HSFLS on the test instances.

6. Conclusion

As its practical importance and complexity, multiobjective flexible job-shop scheduling problem (FJSP) has received considerable attention during last few years, and various heuristic or meta-heuristic approaches were reported in the literature. In this paper, we present a multiobjective evolutionary approach for solving the problem. In our algorithm, a permutation based chromosome representation and several efficient genetic operators are designed for FJSP. For multiobjective optimization problems, an important issue is obtaining well spread nondominated solutions. In other words, good diversity should be maintained in the population. To do this, a modified crowding distance measure is introduced in our approach to indicate the diversity of individuals in decision space. In order to speed up

the convergency of the algorithm, a local search procedure is embedded in the evolutionary process. Experiment results indicate the effectiveness of the proposed approach. In order to handle more complex problems, in future work, the authors will address the design of more efficient hybrid multiobjective evolutionary algorithms, in which several mentioned issues, that is, balance between genetic search and local search and choice of solutions to which local search is applied, will be considered in the implementation of algorithms. In this research, it is assumed that all parameters and data are deterministic. However, in real applications, schedules are often confronted with uncertain factors. In other words, the executing environment of a schedule might be stochastic or dynamic. The authors will address the efficient approach for stochastic version of FJSP in future work.

Acknowledgments

This research was supported in part by the China Scholarship Council and National Natural Science Foundation of China under the Contracts no. 70971131, 70901074, 71001104, 71101150, and 71101096.

References

[1] M. R. Garey, D. S. Johnson, and R. Sethi, "The complexity of flowshop and job shop scheduling," *Mathematics of Operations Research*, vol. 1, no. 2, pp. 117–129, 1976.
[2] A. S. Jain and S. Meeran, "Deterministic job-shop scheduling: past, present and future," *European Journal of Operational Research*, vol. 113, no. 2, pp. 390–434, 1999.
[3] P. Brucker and R. Schlie, "Job-shop scheduling with multi-purpose machines," *Computing*, vol. 45, no. 4, pp. 369–375, 1990.
[4] I. Kacem, S. Hammadi, and P. Borne, "Approach by localization and multiobjective evolutionary optimization for flexible job-shop scheduling problems," *IEEE Transactions on Systems, Man and Cybernetics Part C*, vol. 32, no. 1, pp. 1–13, 2002.
[5] 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. 3–5, pp. 245–276, 2002.
[6] W. Xia and Z. Wu, "An effective hybrid optimization approach for multi-objective flexible job-shop scheduling problems," *Computers and Industrial Engineering*, vol. 48, no. 2, pp. 409–425, 2005.
[7] T. Hsu, R. Dupas, D. Jolly, and G. Goncalves, "Evaluation of mutation heuristics for the solving of multiobjective flexible job shop by an evolutionary algorithm," in *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics*, pp. 655–660, October 2002.
[8] 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 and Industrial Engineering*, vol. 56, no. 4, pp. 1309–1318, 2009.
[9] H. Liu, A. Abraham, and Z. Wang, "A multi-swarm approach to multi-objective flexible job-shop scheduling problems," *Fundamenta Informaticae*, vol. 95, no. 4, pp. 465–489, 2009.
[10] 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 and Industrial Engineering*, vol. 53, no. 1, pp. 149–162, 2007.
[11] J. Gao, L. Sun, and M. Gen, "A hybrid genetic and variable neighborhood descent algorithm for flexible job shop scheduling problems," *Computers and Operations Research*, vol. 35, no. 9, pp. 2892–2907, 2008.
[12] L. N. Xing, Y. W. Chen, and K. W. Yang, "Multi-objective flexible job shop schedule: design and evaluation by simulation modeling," *Applied Soft Computing Journal*, vol. 9, no. 1, pp. 362–376, 2009.
[13] L. N. Xing, Y. W. Chen, and K. W. 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.
[14] J. Q. Li, Q. K. Pan, and Y. C. Liang, "An effective hybrid tabu search algorithm for multi-objective flexible job-shop scheduling problems," *Computers and Industrial Engineering*, vol. 59, no. 4, pp. 647–662, 2010.

[15] A. Bagheri, M. Zandieh, I. Mahdavi, and M. Yazdani, "An artificial immune algorithm for the flexible job-shop scheduling problem," *Future Generation Computer Systems*, vol. 26, no. 4, pp. 533–541, 2010.

[16] G. Vilcot and J.-C. Billaut, "A tabu search algorithm for solving a multicriteria flexible job shop scheduling problem," *International Journal of Production Research*, vol. 49, no. 23, pp. 6963–6980, 2011.

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

[18] M. Frutos, A. C. 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. 5–8, pp. 757–767, 2010.

[21] J. Q. Li, Q. K. Pan, and K. Z. 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. 9–12, pp. 1159–1169, 2011.

[22] J.-Q. Li, Q.-K. 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.

[23] 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.

[24] E. Balas, "Machine sequencing via disjunctive graphs: an implicit enumeration algorithm," *Operations Research*, vol. 17, pp. 941–957, 1969.

[25] K. Deb and S. Tiwari, "Omni-optimizer: a generic evolutionary algorithm for single and multi-objective optimization," *European Journal of Operational Research*, vol. 185, no. 3, pp. 1062–1087, 2008.

[26] T. Ulrich, J. Bader, and E. Zitzler, "Integrating decision space diversity into hypervolume-based multiobjective search," in *Proceedings of the 12th Annual Genetic and Evolutionary Computation Conference (GECCO '10)*, pp. 455–462, Portland, Ore, USA, July 2010.

[27] H. Ishibuchi, N. Akedo, and Y. Nojima, "A many-objective test problem for visually examining diversity maintenance behavior in a decision space," in *Proceedings of the 13th Annual Genetic and Evolutionary Computation Conference (GECCO '11)*, pp. 649–656, 2011.

[28] M. Gen, Y. Tsujimura, and E. Kubota, "Solving job-shop scheduling problems by genetic algorithm," in *Proceedings of the 16th International Conference on Computer and Industrial Engineering*, pp. 1577–1582, Ashikaga, Japan, October 1994.

[29] F. Pezzella, G. Morganti, and G. Ciaschetti, "A genetic algorithm for the Flexible Job-shop Scheduling Problem," *Computers and Operations Research*, vol. 35, no. 10, pp. 3202–3212, 2008.

[30] S. Hartmann and R. Kolisch, "Experimental evaluation of state-of-the-art heuristics for the resource-constrained project scheduling problem," *European Journal of Operational Research*, vol. 127, no. 2, pp. 394–407, 2000.

[31] R. Kolisch and S. Hartmann, "Experimental investigation of heuristics for resource-constrained project scheduling: an update," *European Journal of Operational Research*, vol. 174, no. 1, pp. 23–37, 2006.

[32] T. Murata, H. Ishibuchi, and H. Tanaka, "Genetic algorithms for flowshop scheduling problems," *Computers and Industrial Engineering*, vol. 30, no. 4, pp. 1061–1071, 1996.

[33] T. Murata, H. Ishibuchi, and H. Tanaka, "Multi-objective genetic algorithm and its applications to flowshop scheduling," *Computers and Industrial Engineering*, vol. 30, no. 4, pp. 957–968, 1996.

[34] J. Xiong, Y.-W. Chen, K.-W. Yang, Q.-S. Zhao, and L.-N. Xing, "A hybrid multiobjective genetic algorithm for robust resource-constrained project scheduling with stochastic durations," *Mathematical Problems in Engineering*, vol. 2012, Article ID 786923, 24 pages, 2012.

[35] S. Hartmann, "A competitive genetic algorithm for resource-constrained project scheduling," *Naval Research Logistics*, vol. 45, no. 7, pp. 733–750, 1998.

[36] S. Shadrokh and F. Kianfar, "A genetic algorithm for resource investment project scheduling problem, tardiness permitted with penalty," *European Journal of Operational Research*, vol. 181, no. 1, pp. 86–101, 2007.

[37] P. Moscato and R. Cheng, "A memetic approach for the traveling salesman problem: implementation of a computational ecology for combinatorial optimization on message-passing systems," in

Proceedings of the International Conference on Parallel Computing and Transputer Applications, Amsterdam, The Netherlands, 1992.

[38] H. Ishibuchi and T. Murata, "Multi-objective genetic local search algorithm," in *Proceedings of the IEEE International Conference on Evolutionary Computation (ICEC '96)*, pp. 119–124, May 1996.

[39] 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*, vol. 28, no. 3, pp. 392–403, 1998.

[40] C. Zhang, P. Li, Z. Guan, and Y. Rao, "A tabu search algorithm with a new neighborhood structure for the job shop scheduling problem," *Computers and Operations Research*, vol. 34, no. 11, pp. 3229–3242, 2007.

[41] 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.

[42] J.-Y. Lin and Y.-P. Chen, "Analysis on the collaboration between global search and local search in memetic computation," *IEEE Transactions on Evolutionary Computation*, vol. 15, no. 5, Article ID 6031912, pp. 608–623, 2011.

[43] H. Ishibuchi, Y. Hitotsuyanagi, Y. Wakamatsu, and Y. Nojima, "How to choose solutions for local search in multiobjective combinatorial memetic algorithms," in *Proceedings of the 11th International Conference on Parallel Problem Solving from Nature (PPSN '10)*, 2010.

[44] N. Karimi, M. Zandieh, and H. R. Karamooz, "Bi-objective group scheduling in hybrid flexible flowshop: a multi-phase approach," *Expert Systems with Applications*, vol. 37, no. 6, pp. 4024–4032, 2010.

[45] S. H. A. 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.

[46] 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.

[47] 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.