A Memetic Algorithm for the Multi-Objective Flexible Job Shop Scheduling Problem

Yuan Yuan, Hua Xu
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, P.R.China
yyxhdy@gmail.com, xuhua@mail.tsinghua.edu.cn

ABSTRACT
In this paper, a new memetic algorithm (MA) is proposed for the multi-objective flexible job shop scheduling problem (MO-FJSP) with the objectives to minimize the makespan, total workload and critical workload. By using well-designed chromosome encoding/decoding scheme and genetic operators, the non-dominated sorting genetic algorithm II (NSGA-II) is first adapted for the MO-FJSP. Then the MA is developed by incorporating a novel local search algorithm into the adapted NSGA-II, where several mechanisms to balance the genetic search and local search are employed. 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. Experimental results on well-known benchmark instances show that the proposed MA outperforms significantly two off-the-shelf multi-objective evolutionary algorithms and four state-of-the-art algorithms specially proposed for the MO-FJSP.

Categories and Subject Descriptors
I.2.8 [Problem Solving, Control Methods, and Search]: Scheduling; Heuristic methods

General Terms
Algorithms

Keywords
Muti-Objective, Flexible job shop scheduling, Memetic algorithm, Non-dominated sorting genetic algorithm II (NSGA-II), Local search

1. 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 an extension 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 [7].

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. Compared to the SO-FJSP, the research on the multi-objective 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 $f_1$, $f_2$, …, $f_n$, a single objective problem is derived with an aggregation 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 major drawback in this approach is that the weight coefficient $w_i$ for each objective must be given a prior. According to the existing literature, earlier research on the MO-FJSP mainly concentrated on this approach. Xia and Wu [21] 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. [16] presented a hybrid meta-heuristic combining PSO and variable neighborhood search (VNS) to solve the MO-FJSP. Gao et al. [6] developed a new genetic algorithm (GA) hybridized with a bottleneck shifting procedure. Zhang et al. [23] 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. [22] 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. The posteriori approach is in fact more desirable, which aims to seek for the set of Pareto optimal solutions. Very recently, to solve the MO-FJSP in this Pareto way has been more concerned by the researchers. Kacem et al. [10] proposed a Pareto approach based on the hybridization of fuzzy logic (FL) and evolutionary algorithms to solve the MO-FJSP. Ho and Tay [8] 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. Wang et al. [20] presented a multi-objective GA based on immune and entropy principle for the MO-FJSP. Mosleh and Mahnam [18] proposed a new approach hybridizing FSO and local search. In [14] and [15], 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 non-dominated sorting genetic algorithm II (NSGA-II) for individual evaluation. Li et al. [13] also proposed a hybrid Pareto-based local search embedding a VNS based self-adaptive strategy for the same problem. Chiang et al. [3] proposed a multi-objective evolutionary algorithm which utilizes effective genetic operators and maintains population diversity carefully. In their subsequent research [2], an effective multi-objective memetic algorithm (MA) was also developed.

Note that all the work on the MO-FJSP mentioned above except for [16] considered the makespan, total workload and critical workload as objectives. While in [16], the MO-FJSP with the makespan and flowtime criteria was studied.

Memetic algorithms (MAs) are a class of techniques that combine evolutionary algorithms with local search, whose success has been demonstrated on a wide variety of real-world problems [11]. In this paper, we adopt the posteriori approach and propose a new MA for the MO-FJSP with the criteria to minimize the makespan, total workload and critical workload. First, a newly adapted NSGA-II for the MO-FJSP is developed through well designed chromosome encoding, chromosome decoding, and genetic operators. Second, a novel problem-specific local search algorithm is proposed. In this local search, 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. Third, our MA is formed by incorporating the proposed local search into the adapted NSGA-II, where several mechanisms to balance the genetic search and local search are employed. Experimental results on well-known benchmark instances show that the proposed MA significantly outperforms two adapted traditional multi-objective evolutionary algorithms and four state-of-the-art algorithms specially proposed for the MO-FJSP.

The rest of this paper is organized as follows. Section 2 formulates the studied problem. The proposed new MA is described in detail in Section 3. Afterwards, experimental results are presented in Section 4. Finally, the paper is summarized in Section 5.

2. PROBLEM FORMULATION

The MO-FJSP can be formulated as the following. There are a set of $n$ independent jobs $J = \{J_1, J_2, \ldots, J_n\}$ 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 $j$th 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:

\[ C_{\text{max}} = \max \{ C_i | i = 1, 2, \ldots, n \} \]  
\[ W_T = \sum_{k=1}^{m} W_k \]  
\[ W_{\text{max}} = \max \{ W_i | k = 1, 2, \ldots, m \} \]

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 1, 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>
<thead>
<tr>
<th>Job</th>
<th>Operation</th>
<th>$M_1$</th>
<th>$M_2$</th>
<th>$M_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$J_1$</td>
<td>$O_{1,1}$</td>
<td>5</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>$J_2$</td>
<td>$O_{2,1}$</td>
<td>3</td>
<td>–</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>$O_{2,2}$</td>
<td>2</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>$O_{2,3}$</td>
<td>–</td>
<td>–</td>
<td>4</td>
</tr>
<tr>
<td>$J_3$</td>
<td>$O_{3,1}$</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>$O_{3,2}$</td>
<td>3</td>
<td>–</td>
<td>2</td>
</tr>
</tbody>
</table>

3. PROPOSED MEMETIC ALGORITHM

3.1 Overview of the MA

The framework of the proposed MA is based on the original NSGA-II [5], and its algorithmic flow 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 maximal number of generations ($Gen_{\text{max}}$) is satisfied. In each generation, 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. In the following subsections, the key components of our MA will be illustrated in detail.

Algorithm 1 Framework of the proposed MA

1: $P_0 \leftarrow $ InitializePopulation()
2: $t \leftarrow 0$
3: while $t < Gen_{\text{max}}$ 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: for $\{F_1, F_2, \ldots\} \leftarrow $ FastNonDominatedSort($R_t$ do
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($P_t$
17: $P_{t+1} \leftarrow P_{t+1} \cup F_i[1 : (N - |P_{t+1}|)]$
18: $t \leftarrow t + 1$
19: end while

3.2 Chromosome Encoding and Decoding

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 MA 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 1, operation 4 has the same reference with operation $O_{2,2}$.

The machine assignment vector, which is denoted by $u = [u_1, u_2, \ldots, u_d]$, 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 $u_j$'th one in its sorted available machines.

The operation sequence vector, $v = [v_1, v_2, \ldots, v_d]$, is an ID permutation of all the operations. The order of occurrence for each operation in the $v$ indicates its scheduling priority. For example, a possible operation sequence vector for the problem shown in Table 1 is represented as $v = [6, 1, 7, 3, 4, 2, 5]$. And it can be directly translated into a unique list of ordered operations: $O_{1,1} \succ O_{1,2} \succ O_{2,1} \succ O_{2,2} \succ O_{1,3} \succ O_{2,3}$. Operation $O_{1,1}$ has the highest priority and is scheduled first, then operation $O_{1,2}$, 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 $v$.

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 $v$. When one operation is treated, its selected machine is first got from the $u$, 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 starting 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 $[S_i, E_s]$ on machine $M_k$ is available for $O_{i,j}$, if

$$\begin{align*}
\max\{S_x, c_{i,j-1}\} + p_{i,j,k} \leq E_s, \quad & \text{if } j \geq 2 \\
S_x + p_{i,j,k} \leq E_s, \quad & \text{if } j = 1
\end{align*}$$

When $O_{i,j}$ is to be allocated in the available interval $[S_i, E_s]$, $\max\{S_x, c_{i,j-1}\}$ ($j \geq 2$) 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 $M_k$. By using this decoding method, a chromosome for the problem shown in Table 1, such as $u = [1, 3, 1, 2, 1, 3, 2]$ and $v = [6, 1, 7, 3, 4, 2, 5]$, is decoded into a schedule represented by the Gantt chart which is illustrated in Fig. 1.

![Gantt chart corresponding to the chromosome.](image)

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 started earlier than $v_j$ in the decoded schedule, while $v_j$ actually appears before $v_i$ in the $v$. To make the operation sequence information well inherited, when a chromosome is decoded, the operations in its $v$ are reordered according to their starting time in the corresponding decoded schedule before it involves genetic operators.

3.3 Genetic Operators

The genetic operators in our MA 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 $u$, a subset of positions are first randomly chosen, then generate the $u$ of children by exchanging the values of these selected positions between two parents. As for the $v$, a modified order crossover [19] 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. 2, the above crossover operator is illustrated for the problem shown in Table 1. The symmetric process is repeated for the second parent and the second child.

Algorithm 2 RepairOperationSequence (v)

1: \[\{q_1, q_2, \ldots, q_d\} \leftarrow [0, 0, \ldots, 0]\]
2: \textbf{for} i = 1 to d \textbf{do}
3: \text{Get the job } J_k \text{ that the operation } v_i \text{ belongs to}
4: \text{ Set } q_k \leftarrow q_k + 1
5: \text{ Get the fixed ID } op \text{ for the operation } O_{v_i,q_k}
6: \text{ Set } v_i \leftarrow op
7: \textbf{end for}

Parent 1
\begin{tabular}{cccccc}
6 & 1 & 7 & 3 & 4 & 2 & 5
\end{tabular}

Child 1
\begin{tabular}{cccccc}
1 & 5 & 7 & 3 & 4 & 2 & 6
\end{tabular}

Parent 2
\begin{tabular}{cccccc}
3 & 4 & 1 & 5 & 2 & 6 & 7
\end{tabular}

Child 1
\begin{tabular}{cccccc}
1 & 3 & 6 & 4 & 5 & 2 & 7
\end{tabular}

Figure 2: Illustration of the crossover for the v.

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

3.4 Local Search Algorithm

There exist several issues to implement the local search (Step 5 in Algorithm 1) in the proposed MA. One is the choice of individuals from the offspring population for the local search, because the applicability of local search to bad solutions is usually waste of computational effort. In our MA, a selection scheme similar to that presented in [9] is adopted. First, the following aggregation function is defined:  
\[ f(x, \lambda) = \lambda_1 f_1(x) + \lambda_2 f_2(x) + \lambda_3 f_3(x) \quad (5) \]

where \(\lambda = [\lambda_1, \lambda_2, \lambda_3]\) is a weight vector, \(f_1(x)\), \(f_2(x)\) and \(f_3(x)\) 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:
\[ \lambda_1 + \lambda_2 + \lambda_3 = z, \quad \lambda_i \in \{0, 1, \ldots, z\}, \ i = 1, 2, 3 \quad (6) \]

In this paper, \(z\) in (6) is set as 13 to produce \(C_{15}^2 = 105\) 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 an improved one, where (5) with the weight vector specified in the selection phase is still used to evaluate the solutions formed in the path of local search. Another issue is how often should local search be applied. We introduce a local search probability \(P_s\) and the number of \([N \times P_s]\) individuals are selected for local search. In other words, the selection of an individual and the applicability of local search are repeated \([N \times P_s]\) times. All the above-mentioned procedures are summarized in Algorithm 3.

Algorithm 3 LocalSearch\((Q_i)\)

1: \(Q'_i \leftarrow \emptyset\)
2: \textbf{for} \(i = 1\) to \([N \times P_s]\) \textbf{do}
3: \text{Randomly draw a weight vector } \lambda \text{ from the weight vector set}
4: \{u, v\} \leftarrow \text{TournamentSelection}(Q_i, \lambda)
5: \{u', v'\} \leftarrow \text{LocalSearchForIndividual}(\{u, v\}, \lambda)
6: \(Q'_i \leftarrow Q'_i \cup \{u', v'\}\)
7: \textbf{end for}
8: \textbf{return} \(Q'_i\)

So far, there is one last important issue remaining to be addressed. That is the local search operator to a chromosome corresponding to Step 5 in Algorithm 3. In our MA, 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.

To clearly describe how our local search to the schedule works, \textit{disjunctive graph}, a kind of representation for the FJSP schedule, is first introduced. Suppose that the disjunctive graph is represented by \(G = (V, C \cup D)\). In the graph, \(V\) denotes a set of all the nodes, each node (excluding starting and ending nodes) represents an operation in the FJSP; \(C\)

Figure 3: Illustration of the disjunctive graph.
is the set of all the conjunctive arcs, these arcs connect two adjacent operations within one job and the directions represent the processing order; $D$ means a set of all the disjunctive arcs, these arcs connect two adjacent operations performed on the same machine and their directions indicate the processing order. The processing time for each operation is labeled below its node and regarded as the weight of the node, while the chosen machine is labeled above. For example, a schedule corresponding to the Gantt chart shown in Fig. 1 is illustrated in Fig. 3 via the disjunctive graph. A schedule is feasible, if and only if its disjunctive graph is acyclic. The longest paths from $S$ to $E$ are called critical paths, whose lengths denote the makespan. Operations on acyclic. The longest paths from

$G$ in Fig. 1 is illustrated in Fig. 3 via the disjunctive graph. The arcs, these arcs connect two adjacent operations within one job and the directions represent the earliest and latest completion time are $ES(G, v)$ and $LS(G, v)$, respectively. These times are calculated by moving an operation. Since the objective makespan is relatively harder to be minimized, we accept the neighbor $G'$ of $G$ only when $C_{max}(G') \leq C_{max}(G)$. Furthermore, the makespan can only be improved by moving critical operations, so only the critical operations are considered to be moved.

For a schedule $G$, denote $\mu(G, v)$ as the selected machine ID for a generic operation $v$ 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 $T$. The corresponding earliest and latest completion time are $EC(G, v) = EC(G, v) + \mu(G, v)$ and $LC(G, v, T) = LS(G, v, T) + \mu(G, v)$. Denote $PM(G, v)$ as the operation that immediately precedes $v$ 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 $\gamma(G) = \{co_1, co_2, \ldots, co_n\}$ be the set of critical operations in $G$, where $n$ is the number of critical operations.

Suppose a critical operation $co_i$ in $G$ is to be moved, first delete it from $G$ to yield $G'_{\neg i}$ 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 $G'_{\neg i}$ to obtain $G''$ so that $C_{max}(G') \leq C_{max}(G)$.

For each such position locate before operation $v$ on machine $M_k$ in $G''$, $co_i$ should be started as early as $EC(G_{\neg i}', PM(G_{\neg i}', v))$, and can be completed as late as $LS(G_{\neg i}', v, C_{max}(G'))$ 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, the following inequality should be satisfied

$$\max\{EC(G_{\neg i}', PM(G_{\neg i}', v)), EC(G_{\neg i}', PJ(co_i))\} + \mu(co_i, k)$$

$$\min\{LS(G_{\neg i}', v, C_{max}(G)), LS(G_{\neg i}', SJ(co_i), C_{max}(G))\}$$

(7)

However, to insert $co_i$ before $v$ under (7) is met can not be ensured the yielded $G''$ is acyclic. Below, the positions to be examined on machine $M_k$ are restricted to only feasible ones. Let $\Theta_k$ be the set of operations processed by machine $M_k$ in $G'_{\neg i}$ and ordered with the increasing of the earliest starting time (note that $co_i \notin \Theta_k$). Denote $\Phi_k$ and $\Psi_k$ as two subsequences of $\Theta_k$ and are defined as follows

$$\Phi_k = \{r \in \Theta_k | ES(G, r, C_{max}(G)) < LS(G, r, co_i)\}$$

(8)

$$\Psi_k = \{r \in \Theta_k | LS(G, r, C_{max}(G)) < LS(G, r, co_i)\}$$

(9)

Let $\Upsilon_k$ 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 1.** In $G'_{\neg i}$, 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 $\Upsilon_k$ 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 theory can be referred in [17]. According to Theorem 1, we have the direct corollary as follows

**Corollary 1.** In $G'_{\neg i}$, 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 $\Upsilon_k$.

With Corollary 1, only the positions in $\Upsilon_k$ are checked when reallocating a position on machine $M_k$ for $co_i$ in $G'_{\neg i}$, once a position satisfying (7) is found, $co_i$ is inserted immediately and a neighbor schedule $G'$ is formed to replace the current schedule $G$. As can be seen, the first acceptance criterion is adopted in our local search, because it is especially computational expensive to check all the positions for all the critical operations. Denote $co_i \rightarrow M_k$ as the action to find a position on machine $M_k$ for $co_i$ to insert into in $G'_{\neg i}$. In summarize, the action $co_i \rightarrow M_k$ is depicted in Algorithm 4.

**Algorithm 4 InsertOperationOnMachine($G'_{\neg i}$, $co_i$, $k$)**

1: Get the set of positions $\Upsilon_k$ on machine $M_k$
2: for each position $\gamma$ in $\Upsilon_k$ do
3: if $\gamma$ satisfies (7) then
4: Insert $co_i$ into $\gamma$ in $G'_{\neg i}$
5: return true
6: end if
7: end for
8: return false

**Algorithm 5 GetNeighborSchedule($G$)**

1: Get the set $\chi(G) = \{co_1, co_2, \ldots, co_n\}$
2: $\Phi_i, \Phi_2, \ldots, \Phi_n \leftarrow \emptyset, \emptyset, \ldots, \emptyset$
3: Sort $\phi(G)$ according to $\Delta_1$ and $\Delta_k$
4: for each action $\phi_i \rightarrow M_k$ in the sorted $\phi(G)$ do
5: if $\Phi_i = \emptyset$ then
6: Clone a copy of $G$ to $G''$
7: Delete $co_i$ from $G''$
8: $G'' \leftarrow G''$
9: end if
10: if InsertOperationOnMachine($G''$, $co_i$, $k$) then
11: $G' \leftarrow G''$
12: return $G'$
13: end if
14: end for
15: return $\emptyset$

Let $\phi(G) = \{co_i \rightarrow M_k | i = 1, 2, \ldots, nc, M_k \in M_{nc}\}$, which consists of all $\sum_{i=1}^{nc}$ possible actions described in Algorithm 4 to form an acceptable neighbor $G'$. 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 \( c_0 \sim M_k \) in \( \varphi(G) \) as follows

\[
\Delta_t(c_0 \sim M_k) = p_{c_0,k} - p_{c_0,M_k(G),G} \quad (10)
\]

\[
\Delta_t(c_0 \sim M_k) = W_t(G) + p_{c_0,k} \quad (11)
\]

Then the actions in \( \varphi(G) \) are sorted according to the non-decreasing order of \( \Delta_t \). If two actions have the same \( \Delta_t \) values, the one with lower \( \Delta_t \) value first. At one iteration of local search, these actions are considered in order until a new neighbor of \( G \) is obtained. This procedure is illustrated in Algorithm 5.

The local search operator to a chromosome is summarized in Algorithm 6, where the parameter \( \text{iter}_{\text{max}} \) is the maximal iterations of local search. The schedule with the lowest \( f(G, \lambda) \) value in the path of local search is encoded and returned as the improved one.

### Algorithm 6: LocalSearchForIndividual(\( \{u, v\}, \lambda \) )

1: \( i \leftarrow 0 \)
2: \( G \leftarrow \text{ChromosomeDecoding}(u, v) \)
3: \( f \leftarrow 0 \)
4: while \( G \neq \emptyset \) and \( i < \text{iter}_{\text{max}} \) do
5: \( G \leftarrow \text{GetNeighborSchedule}(G) \)
6: if \( G \neq \emptyset \) and \( f(G, \lambda) < f(G_{\text{best}}, \lambda) \) then
7: \( G_{\text{best}} \leftarrow G \)
8: \( f \leftarrow i \)
9: end if
10: \( i \leftarrow i + 1 \)
11: end while
12: if \( f \neq 1 \) then
13: \( \{u, v\} \leftarrow \text{ChromosomeEncoding}(G_{\text{best}}) \)
14: return \( \{u, v\} \)
15: end if
16: return \( \emptyset \)

### 4. EXPERIMENTAL RESULTS

The proposed MA is implemented in Java and run on an Intel 2.83GHz Xeon processor with 15.9Gb of RAM. The MA is tested on two sets of the most commonly used benchmarks including five Kacem instances [10] and ten BRdata instances [1]. To show the superiority of the MA, we compare it with two traditional multi-objective evolutionary algorithms, namely NSGA-II and multi-objective evolutionary algorithm based on decomposition (MOEA/D) [12], and four state-of-the-art algorithms specifically proposed for the MO-FJSP. For applying NSGA-II and MOEA/D to the MOFJSP, the chromosome representation and genetic operators of the proposed MA are directly used in both of them. The parameter settings of MA, NSGA-II, and MOEA/D are listed in Table 2. They are set in such a way that the three algorithms share most of important parameters. Because NSGA-II and MOEA/D do not employ the local search process, they are assigned a larger generation number to have a fair comparison with the MA. For each problem instance, all the three algorithms are run 30 times independently.

#### 4.1 Performance Measures

In order to evaluate the performance of the algorithms, the inverted generational distance (IGD) [4] and the set coverage [24] are used as indicators in our experiments. They can be expressed as follows:

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 measure IGD of the set \( A \) is defined as:

\[
\text{IGD}(A, P^*) = \frac{1}{|P^*|} \sum_{x \in P^*} \min_{y \in A} d(x, y) \quad (12)
\]

where \( d(x, y) \) is the Euclidean distance between the points \( x \) and \( y \). If \( P^* \) is large enough to represent the PF, 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 \( P^* \).

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.

\[
C(A,B) = \frac{|\{ x \in B : \exists y \in A : y \text{ dominates } x \}|}{|B|} \quad (13)
\]

\( C(A,B) \) 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 alternatively formed for each problem instance by gathering all non-dominated solutions found by all the three implemented algorithms (MA, NSGA-II, and MOEA/D) in all runs, and the non-dominated solutions obtained by the algorithms proposed in [15], [13], [3], and [2] are also included. These \( P^* \) sets together with the detail computational results in this paper are available on the website\(^3\) for future use of other researchers.

#### 4.2 Comparison with NSGA-II and MOEA/D

The performance of the proposed MA is first compared with the adapted NSGA-II and MOEA/D. Table 3 presents the average and standard deviation of IGD values obtained by the three algorithms for 15 problem instances. From Table 3, it can be seen that MA achieves the best results for all the instances, while NSGA-II only obtains the best for the instances Kacem 4×5 and Mk08. The results obtained by MOEA/D are relative poor, which are worse than those of MA and NSGA-II on all the instances.

Table 4 shows the results of set coverage metric. It is clear from the results in Table 4 that MA outperforms NSGA-II for all the instances but Kacem 4×5 and Mk08, on which MA and NSGA-II obtain the same results. MOEA/D is obviously outperformed again by MA in terms of set coverage.

---

\(^3\)http://166.111.4.17:8080/2012310563/GECCO2013-MOFJSP-Results.rar
and only MA can achieve the best for five of them. MA is for 14 out of 15 problem instances among all algorithms, as can be seen from Table 5, MA produces the lowest IGD values of solutions obtained in all runs for each algorithm. As can be seen, the statistical comparison on IGD or set coverage metric. Only for the instance Mk08, there is no performance difference between MOEA/D and MA from the view of set coverage.

In summary, MA outperforms NSGA-II and MOEA/D on the original NSGA-II, the superior performance of MA compared with NSGA-II also demonstrates the effectiveness of the proposed local search.

4.3 Comparison with State-of-the-Art Algorithms

In this subsection, our MA is evaluated by comparing with four state-of-the-art algorithms recently proposed for the MO-FJSP, which are called HSFLA [15], PLS [13], SEA [3], and CMA [2] respectively in this paper. HSFLA and PLS only considered a subset of all 15 problem instances. In the MO-FJSP literature, the performance of an algorithm is generally presented by listing the non-dominated solutions found over a certain number of runs. However, to our knowledge, there is hardly any algorithm for the MO-FJSP that reports the non-dominated solutions obtained for each run. Hence, the statistical comparison on IGD or set coverage metric between MA and the referred four algorithms seems to be impossible, although we think that this kind of comparison is more justified than the comparison of non-dominated solutions obtained over several runs.

In Table 5, IGD value is computed for the combined set of solutions obtained in all runs for each algorithm. As can be seen from Table 5, MA produces the lowest IGD values for 14 out of 15 problem instances among all algorithms, and only MA can achieve the best for five of them. MA is just slightly outperformed by CMA on the instance Mk08. However, we note that the results of CMA are obtained with four variants of parameter settings, while the parameters of MA are fixed.

Due to space limitation, we do not list the solutions obtained for each instance. Instead, we summarize in Table 6 the size of $P^*$, as well as the number of solutions and the number of non-dominated ones in $P^*$ among these solutions found by each algorithm over all runs. From Table 6, the results of MA are by far the best among all algorithms. Indeed, all solutions in $P^*$ for 12 out of 15 instances can be found by MA. For the instance Mk09, MA misses only one solution. For the instances Mk06 and Mk10, MA obtains about 74% and 60% of solutions in $P^*$ respectively, while CMA only achieves about 38% and 16%. We recall that CMA obtains the lowest IGD value on the instance Mk06, it could be explained that the solutions obtained by CMA on this problem exhibit more diversity, while those by MA present more proximity to PF. Moreover, we observe that those non-dominated solutions that are not found by MA usually have a relatively low critical workload value. One possible reason is that this objective is given the least consideration in the local search process.

5. CONCLUSION AND FUTURE WORK

This paper presents a new MA to solve the MO-FJSP with the makespan, total workload, and critical workload criteria, where a novel local search algorithm is integrated into the adapted NSGA-II framework. The local search in fact adopts a hierarchical strategy to deal with the three objectives. That is, the non-increasing of makespan is considered 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. Experimental results on two well-known benchmark sets demonstrate the advantages of the proposed MA over two adapted multi-objective evolutionary algorithms and four state-of-the-art algorithms specially proposed for MO-FJSP. The success of our MA verifies the effectiveness of combining traditional multi-objective evolutionary techniques with problem-specific search algorithms. In future, we will further improve the performance of our MA by using self-adaptive strategies to control the probability of local search.

Acknowledgment

This work is 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).

6. REFERENCES