A tailored NSGA-III for multi-objective flexible job shop scheduling

Wang, Y.; Stein, N. van; Bäck, T.H.W.; Emmerich, M.T.M.

A Tailored NSGA-III for Multi-objective Flexible Job Shop Scheduling

1st Yali Wang
LIACS, Leiden University
Leiden, The Netherlands
y.wang@liacs.leidenuniv.nl

2nd Bas van Stein
LIACS, Leiden University
Leiden, The Netherlands
b.van.stein@liacs.leidenuniv.nl

3rd Thomas Bäck
LIACS, Leiden University
Leiden, The Netherlands
t.h.w.baeck@liacs.leidenuniv.nl

4th Michael Emmerich
LIACS, Leiden University
Leiden, The Netherlands
m.t.m.emmerich@liacs.leidenuniv.nl

***Abstract*—A customized multi-objective evolutionary algorithm (MOEA) is proposed for the flexible job shop scheduling problem (FJSP) with three objectives: makespan, total workload, critical workload. In general, the algorithm can be integrated with any standard MOEA. In this paper, it has been combined with NSGA-III to solve the state-of-the-art benchmark FJSPs, whereas an off-the-shelf implementation of NSGA-III is not capable of solving them. Most importantly, we use the various algorithm adaptations to enhance the performance of our algorithm. To be specific, it uses smart initialization approaches to enrich the first-generation population, and proposes new crossover operator to create a better diversity on the Pareto front approximation. The *MIP-EGO configurator* is adopted to automatically tune the mutation probabilities, which are important hyper-parameters of the algorithm. Furthermore, different local search strategies are employed to explore the neighborhood for better solutions. The experimental results from the combination of these techniques show the good performance as compared to classical evolutionary scheduling algorithms and it requires less computing budget. Even some previously unknown non-dominated solutions for the *BRdata* benchmark problems could be discovered.**

***Index Terms*—Flexible job shop scheduling, Multi-objective optimization, Evolutionary algorithm.**

I. Introduction

The Job shop scheduling problem (JSP) is an important branch of production planning problems. The classical JSP consists of a set of independent jobs to be processed on multiple machines and each job contains a number of operations with a predetermined order. It is assumed that each operation must be processed on a specific machine with a specified processing time. The JSP is to determine a schedule of jobs, meaning to sequence operations on the machines. The flexible job shop scheduling problem (FJSP) is an important extension of the classical JSP due to the wide employment of multi-purpose machines in the real-world job shop. The FJSP extends the JSP by assuming that each operation is allowed to be processed on a machine out of a set of alternatives, rather than one specified machine. Therefore, the FJSP is not only to find the best sequence of operations on a machine, but also to assign each operation to a machine out of a set of qualified machines.

In this paper, evolutionary algorithms (EAs) have been applied to solve a multi-objective flexible job shop scheduling problem (MOFJSP). Due to the NP-hard nature of these problems [1], it is hardly possible to find their true Pareto fronts (PFs) and a basic EA can not perform well on the MOFJSPs. Therefore, we develop some techniques and use them together with EA to enhance the performance. Firstly, we propose and adopt multiple initialization approaches to produce the first-generation population based on our definition of the chromosome representation. In this way, the algorithm can produce a better-distributed initial population which can increase the robustness and avoid premature convergence. Secondly, diverse genetic operators are applied to guide the search towards offspring with a wide diversity. Moreover, to find the best parameter setting, we use an algorithm configurator, i.e., MIP-EGO [2], which is usually used in the region of machine learning to optimize the hyper-parameters and neural network architecture. In our algorithm, we apply it to tune the operator probability. Furthermore, two levels of local search are employed for making our algorithm converge faster to the PF. Although it is difficult to implement a local search in a multi-objective context, our implementation of local search is straightforward, avoids getting stuck in local optima and can guarantee the generation of better solutions. These techniques can improve the algorithm individually, but when they work together, the overall hybrid approach reveals a best performance. The experimental results show that our algorithm can achieve state-of-the-art results with less computing effort when it is merged with NSGA-III [3].

The paper is organized as follows. The next section formulates the MOFJSP. Section III gives necessary background knowledge. Section IV introduces the proposed algorithm and Section V reports the experimental results. Finally, Section VI concludes the work and suggests future work directions.

II. PROBLEM FORMULATION

The MOFJSP addressed in this paper is described as:

1) There are n jobs $J = \{J_1, J_2, \cdots, J_n\}$ and m machines $M = \{M_1, M_2, \cdots, M_m\}$.
2) Each job J_i comprises l_i operations for $i = 1, \cdots, n$, the jth operation of job J_i is represented by O_{ij}, and the operation sequence of job J_i is from O_{i1} to O_{il_i}.
3) For each operation O_{ij}, there is a set of machines capable of performing it, which is represented by M_{ij} and it is a subset of M.

978-1-7281-2547-3/20/$31.00 ©2020 IEEE

4) The processing time of the operation O_{ij} on machine M_k is predefined and denoted by t_{ijk}.

At the same time, the following assumptions are made:

1) All machines and jobs are available at time 0 and assumed to be continuously available.
2) All jobs are independent from each other.
3) Setting up times of machines and transportation times between operations are negligible.
4) A machine can only work on one operation at a time.
5) There are no precedence constraints among the operations of different jobs, and the order of operations for each job cannot be modified.
6) An operation, once started, must run to completion.
7) No operation for a job can be started until the previous operation for that job is completed.

The makespan, total workload and critical workload, which are commonly considered in the literature on FJSP and very practical in real-world circumstances, are minimized and used as three objectives in our algorithm. The makespan is defined as the maximum time for completion of all jobs, in other words, the total length of the schedule. Minimizing the makespan can facilitate the rapid response to the market demand. The total workload represents the total working time of all machines and the critical workload is the maximum workload among all machines. Minimizing the total workload can reduce the use of machines; minimizing the critical workload can balance the workload between machines. Let C_i denote the completion time of job J_i, W_k the sum of processing time of all operations that are processed on machine M_k. The three objectives can be defined as follows:

$$\text{Makespan}(C_{max}) : f_1 = \max\{C_i | i = 1, 2, \cdots, n\} \tag{1}$$

$$\text{Total workload}(W_t) : f_2 = \sum_{k=1}^{m} W_k \tag{2}$$

$$\text{Critical workload}(W_{max}) : f_3 = \max\{W_k | k = 1, 2, \cdots, m\} \tag{3}$$

TABLE I
PROCESSING TIME OF A FJSP INSTANCE

Job	Operation	M_1	M_2	M_3
J_1	O_{11}	3	-	2
	O_{12}	5	7	6
	O_{13}	-	-	2
J_2	O_{21}	2	4	3
	O_{22}	2	-	1
J_3	O_{31}	4	2	2
	O_{32}	3	5	-

An example of the MOFJSP is shown in Table I as an illustration, where rows correspond to operations and columns correspond to machines. In this example, there are three machines: M_1, M_2 and M_3. Each entry of the table denotes the processing time of that operation on the corresponding machine, and the tag "$-$" means that a machine cannot execute the corresponding operation.

III. RELATED WORK

A. Algorithms for MOFJSP

According to [4], EA is the most popular non-hybrid technique to solve the FJSP. Among all EAs for FJSP, some are developed for the more challenging FJSP: the MOFJSP which we formulated in Section II. [5], [6] and [7] are very successful MOFJSP algorithms and have obtained high-quality solutions. [5] proposes a multi-objective genetic algorithm (MOGA) based on the immune and entropy principle. In this MOGA, the fitness is determined by the Pareto dominance relation and the diversity is kept by the immune and entropy principle. In [6], a simple EA (SEA) is proposed, which uses domain heuristics to generate the initial population and balance the exploration and exploitation by refining duplicate individuals with mutation operators. A memetic algorithm (MA) is proposed in [7] and it incorporates a local search into NSGA-II. A hierarchical strategy is adopted in the local search to handle three objectives. In Section V, these algorithms have been compared with our algorithm on the MOFJSP.

B. Parameter Tuning

EA involves using multiple parameters and their preset values affect the performance of the algorithm in different situations. These parameters are usually set to values which are assumed to be good. For example, the mutation probability normally is kept very low, otherwise the convergence is supposed to be delayed unnecessarily. But the best way to identify the probability would be to do a sensitivity analysis: carrying out multiple runs of the algorithms with different mutation probabilities and comparing the outcomes. Although there are some self-tuning techniques for adjusting these parameters on the go, the parameters in EA can be optimized using techniques from machine learning.

The optimization of hyper-parameters and neural network architectures is an essential topic in the field of machine learning due to the large number of design choices for a network architecture and its parameters. Recently, algorithms have been developed to accomplish this challenge automatically since it is intractable to do it by hand. The MIP-EGO is one of these configurators that can automatically configure convolutional neural network architectures and the resulting optimized neural networks have been proven to be competitive with the state-of-the-art manually designed ones on some popular classification tasks. Moreover, MIP-EGO allows for multiple candidate points to be selected and evaluated in parallel, which can speed up the automatic tuning procedure. In our paper, we choose MIP-EGO to tune the parameter values and find the best parameter setting.

C. NSGA-III

NSGA-III is a decomposition-based MOEA, it is an extension of the well-known NSGA-II and eliminates the drawbacks of NSGA-II such as the lack of uniform diversity among a set of non-dominated solutions. The basic framework of NSGA-III is similar to the original NSGA-II, while it replaces the crowding distance operator with a clustering operator based on

a set of reference points. A widely-distributed set of reference points can efficiently promote the population diversity during the search and NSGA-III defines a set of reference points by Das and Dennis′s method [8].

In each iteration t, an offspring population Q_t of size N_{pop} is created from the parent population P_t of size N_{pop} using usual selection, crossover and mutation. Then a combined population $R_t = P_t \cup Q_t$ is formed and classified into different layers (F_1, F_2, and so on), each layer consists of mutually non-dominated solutions. Thereafter, starting from the first layer, points are put into a new population S_t. A whole population is obtained until the first time the size of S_t is equal to or larger than N_{pop}. Suppose the last layer included in S_t is the l-th layer, so far, members in $S_t \setminus F_l$ are points that have been chosen for P_{t+1} and the next step is to choose the remaining points from F_l to make a complete P_{t+1}. In general (when the size of S_t doesn't equal to N_{pop}), $N_{pop} - |S_t \setminus F_l|$ solutions from F_l needs to be selected for P_{t+1}.

When selecting individuals from F_l, first, each member in S_t is associated with a reference point by searching the shortest perpendicular distance from the member to all reference lines created by joining the ideal point with reference points. Next, a niching strategy is employed to choose points associated with the least reference points in P_{t+1} from F_l. The niche count for each reference point, defined as the number of members in $S_t \setminus F_l$ that are associated with the reference point, is computed. The member in F_l associated with the reference point having the minimum niche count is included in P_{t+1}. The niche count of that reference point is then increased by one and the procedure is repeated to fill the remaining population slots of P_{t+1}.

NSGA-III is powerful to handle problems with non-linear characteristics as well as having many objectives. Therefore, we decided to combine NSGA-III in our algorithm for the MOFJSP.

IV. PROPOSED ALGORITHM

The proposed algorithm, *Flexible Job Shop Problem Multi-objective Evolutionary Algorithm* (FJSP-MOEA) can in principal be combined with any MOEA to solve the MOFJSP, whereas the standard MOEAs cannot solve MOFJSP solely. The algorithm follows the flow of a typical EA and generates improved solutions by using local search. Details of the components of the proposed FJSP-MOEA are given in the following subsections.

A. Chromosome Encoding

The MOFJSP is a combination of assigning each operation to a machine and ordering operations on the machines. In the algorithm, each chromosome (individual) represents a solution in the search space and the chromosome consists of two parts: the operation sequence vector and the machine assignment vector. Let N denote the number of all operations of all jobs. The length of both vectors is equal to N. The operation sequence vector decides the sequence of operations assigned to each machine. For any two operations which are processed by the same machine, the one located in front is processed earlier than the other one. The machine assignment vector assigns the operations to machines, in other words, it determines which operation is processed by which machine and the machine should be the one capable of processing the operation.

The format of representing an individual not only influences the implementation of crossover and mutation operators, a proper representation can also avoid the production of infeasible solutions and reduces the computational time. In our algorithm, the chromosomal representation proposed by Zhang et al. in [9] is adopted and an example is given in Table II. In the table, the first row shows the operation sequence vector which consists of only job indexes. For each job, the first appearance of its index represents the first operation of that job and the second appearance of the same index represents the second operation of that job, and so on. The occurrence number of an index is equal to the number of operations of the corresponding job. The second row explains the first row by giving the real operations. The third row is the machine assignment vector which presents the selected machines for operations. The operation sequence of the machine assignment vector is fixed, which is from the first job to the last job and from the first operation to the last operation for each job. The fourth row indicates the fixed operation sequence of the machine assignment vector and the fifth row shows the real machines of the operations. Each integer value in the machine assignment vector is the index of the machine in the set of alternative machines of that operation. In this example, O_{13} is assigned to M_3 because M_3 is the first (and only) machine in the alternative machine set of O_{13} (Table I). The alternative machine set of O_{22} is $\{M_1, M_3\}$, the second machine in this set is M_3, therefore, O_{22} is assigned to M_3.

TABLE II
AN EXAMPLE OF THE CHROMOSOME REPRESENTATION

Operation sequence	**1**	**2**	**3**	**2**	**1**	**1**	**3**
	O_{11}	O_{21}	O_{31}	O_{22}	O_{12}	O_{13}	O_{32}
Machine assignment	**2**	**1**	**1**	**3**	**2**	**2**	**1**
	O_{11}	O_{12}	O_{13}	O_{21}	O_{22}	O_{31}	O_{32}
	M_3	M_1	M_3	M_3	M_3	M_2	M_1

B. Population Initialization

Our algorithm starts by creating the initial population. The machine assignment and operation sequence vectors are generated separately for each individual. In the literature, a few approaches have been proposed for producing individuals, such as global minimal workload [10]; AssignmentRule1 and AssignmentRule2 [11]. In our algorithm, we propose *Processing Time Roulette Wheel* (PRW) and *Workload Roulette Wheel* (WRW) for initializing the machine assignment vector; we also propose *Most Remaining Machine Operations* (MRMO) and *Most Remaining Machine Workload* (MRMW) for initializing the operation sequence vector.

PRW and WRW are proposed to assign the operation to the machine with less processing time or accumulated workload, at the same time, maintain the freedom of exploring the entire search space. MRMO and MRMW are proposed to give priority to both the machine and the job with the most number

of remaining operations and the longest remaining processing time. These new approaches have been used together with some commonly used dispatching rules in initializing individuals on the purpose of enriching the initial population. The complete initialization methods used in our algorithm are introduced in the following lists. When generating a new individual in our algorithm, two initialization methods are randomly picked: one for the machine assignment vector and one for the operation sequence vector.

Initialization methods for machine assignment

1) Random assignment (Random): an operation is assigned to an eligible machine randomly.
2) Processing time Roulette Wheel (PRW): for each operation, the roulette wheel selection is adopted to select a machine from its machine set based on the processing times of these capable machines. The machine with the shorter processing time is more likely to be selected.
3) Workload Roulette Wheel (WRW): for each operation, the roulette wheel selection is used to select a machine from its machine set based on the current workloads plus the processing times of these capable machines. The machine with lower sum of the workload and processing time is more likely to be selected.

Initialization methods for operation sequence

1) Random permutation (Random): starting from a fixed sequence: all job indexes of J_1 (the number of J_1 job indexes is the number of operations of J_1), followed by all job indexes of J_2, and so on. Then the array with the fixed sequence is permuted and a random order is generated.
2) Most Work Remaining (MWR): operations are placed one by one into the operation sequence vector. Before selecting an operation, the remaining processing times of all jobs are calculated respectively, the first optional operation of the job with the longest remaining processing time is placed into the chromosome.
3) Most number of Operations Remaining (MOR): operations are placed one by one into the operation sequence vector. Before selecting an operation, the number of succeeding operations of all jobs is counted respectively, the first optional operation of the job with the most remaining operations is placed into the chromosome.
4) Long Processing Time (LPT) [12]: operations are placed one by one into the operation sequence vector, each time, the operation with maximal processing time is selected without breaking the order of jobs.
5) Most Remaining Machine Operations (MRMO): operations are placed into the operation sequence vector according to both the number of subsequent operations on machines and the number of subsequent operations of jobs. MRMO is a hierarchical method and takes the machine assignment into consideration. First, the machine with the most subsequent operations is selected. After that, the optional operations in the subsequent operations on that machine are found based on the already placed operations. For example, if $O_{11} \rightarrow O_{12} \rightarrow O_{21}$ are placed operations, the current optional operation can only be chosen from O_{13}, O_{22}, and O_{31}. In these optional operations, those which are assigned to the selected machine are picked and the one that belongs to the job with the most subsequent operations is placed into the chromosome. In this example, O_{31} will be chosen if it is assigned to the selected machine because there are two subsequent operations for J_3 and only one subsequent operation for J_1 and J_2. Note that it is possible that no operation is available on that machine, in that case, the machine with the second biggest number of subsequent operations will be selected, and so forth.
6) Most Remaining Machine Workload (MRMW): operations are placed into the operation sequence vector according to the remaining processing times of machines and the remaining processing times of jobs. MRMW is a hierarchical method similar to MRMO. After finding the machine with the longest remaining process time and the optional operations on that machine, the operation which belongs to the job with the longest remaining process time is placed into the chromosome. Again, if no operation is available on that machine, the machine with the second longest remaining processing time will be selected, and so forth.

C. Crossover

Crossover is a matter of replacing some of the genes in one parent with the corresponding genes of the other (Glover and Kochenberger [13]). Since our representation of chromosomes has two parts, crossover operators applied to these two parts of chromosomes are implemented separately as well. We propose the new crossover operator for operation sequence, *Precedence Preserving Two Points Crossover* (PPTP), and used it together with several other operators from literature. When executing the crossover operation in the proposed algorithm, one crossover operator for machine assignment and one operator for the operation sequence, are randomly chosen from the following two lists to generate the offspring.

Crossover operators for machine assignment

1) No crossover
2) One point crossover: a cutting point is picked randomly and genes after the cutting point are swapped between two parents.
3) Two points crossover: two cutting points are picked randomly and genes between the two points are swapped between two parents.
4) Job-based crossover (JX):
   a A vector with the size of the jobs is generated, which consists of random values 0 and 1.
   b For the job corresponding to value 0, the assigned machines of its operations are preserved.
   c For the job corresponding to value 1, the machines of its operations are swapped between two parents.
5) Multi-point preservative crossover (MPX) [14]:
   a A vector with the size of all operations is generated, which consists of random values 0 and 1.

b For the operations corresponding to value 0, their machines (genes) are preserved.
c For the operations corresponding to value 1, their machines (genes) are swapped between the two parents.

Crossover operators for operation sequence

1) No crossover
2) Precedence preserving one point crossover (PPOP) [16]:
   a A cutting point is picked randomly, genes to the left are preserved and copied from parent1 to child1 and from parent2 to child2.
   b The remaining operations in parent1 are reallocated in the order they appear in parent2.
   c The remaining operations in parent2 are reallocated in the order they appear in parent1.

   An example of PPOP is shown in Figure 1 and the cutting point is between the third and fourth operation. Red numbers in parent2 are the genes on the right side of the cutting point in parent1 and they are copied to child1 with their own sequence following the genes on the left side of the cutting point in parent1, and vice versa.

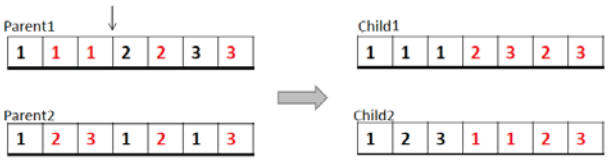

Fig. 1. The process of PPOP

3) Precedence Preserving Two Points Crossover (PPTP):
   a Two cutting points are picked randomly, genes except for those between the two points are preserved and copied from parent1 to child1 and from parent2 to child2.
   b Operations between the two cutting points in parent1 are reallocated in the order they appear in parent2.
   c Operations between the two cutting points in parent2 are reallocated in the order they appear in parent1.
4) Improved precedence operation crossover (IPOX) [15]: IPOX divides the job set into two complementary and non-empty subsets randomly. The operations of one job subset are preserved, while the operations of another job subset are copied from another parent.
5) Uniform Preservative crossover (UPX):
   a A vector with the size of all operations is generated, which consists of random values 0 and 1.
   b For the operations corresponding to value 0, the genes are preserved and copied from parent1 to child1 and from parent2 to child2.
   c For the operations corresponding to value 1, the genes in parent1 are found in parent2 and copied from parent2 with the sequence in parent2, and vice versa.

D. Mutation

The mutation operator flips the gene values at selected locations. In our algorithm, insertion mutation and swap mutation (including one point swap and two points swap) are proposed to generate a new individual by the following procedures:

Insertion Mutation Operator:

a Two random numbers i and j ($1 \leq i \leq N$, $1 \leq j \leq N$) are selected.
b For the operation sequence vector, the operation on position j is inserted in front of the operation on i.
c For the machine assignment vector, a machine is randomly selected for both the operations on i and on j respectively. If the processing time on the newly selected machine is lower than that on the current machine, the current machine is replaced by the new machine. If the processing time on the new machine is longer than that on the old machine, there is only a 20% probability that the new machine replaces the old machine.

Swap Mutation Operator:

a One random number i ($1 \leq i \leq N$) is selected or two random numbers i and j ($1 \leq i \leq N$, $1 \leq j \leq N$) are selected.
b For the operation sequence vector, with only one swap point i, the operation on the swap point is swapped with its neighbour; with two swap points, the operations on position i and j are swapped.
c For the machine assignment vector, the machine on position i is replaced with a new machine by the same rule used in the insertion mutation operator. (For two swap points, both machines are replaced.)

It is worth mentioning that the new solutions (offspring) generated by our crossover and mutation operators are always feasible because these operators are designed based on the adopted definition of the chromosome representation.

E. Decoding and Local Search

Decoding a chromosome is to convert an individual into a feasible schedule to calculate the objective values which represent the relative superiority of a solution. In this process, the operations are picked one by one from the operation sequence vector and placed on the machines from the machine assignment vector to form the schedule. When placing each operation to its machine, local search (in the sense of heuristic rules to improve solution) is involved to refine an individual in order to obtain an improved schedule in the proposed algorithm. We know that idle times may exist between operations on each machine due to precedence constraints among operations of each job. The following two levels of local search are applied to allocate each operation to an idle time slot on its machine and they utilize idle times in different degrees.

The first level local search: let S_{ij} be the starting time of O_{ij} and C_{ij} the completion time of O_{ij}, an example of the first level local search is shown in Figure 2. Because O_{mn} needs to be processed after the completion of O_{mn-1}, an idle time interval between the completion of O_{ab} and the starting of O_{mn} appeared on machine M_k. O_{ij} is assigned to M_k and we assume that O_{mn} is the last operation on M_k before handling O_{ij}, therefore the starting time of O_{ij} is $\max\{C_{mn}, C_{ij-1}\}$, which in this example is C_{mn} and it is later than C_{ij-1}, thus, there is an opportunity that O_{ij} can be processed earlier.

When checking the idle time on M_k, the idle time interval $[C_{ab}, S_{mn}]$ is found available for O_{ij} because the idle time span $[C_{ij-1}, S_{mn}]$, which is part of $[C_{ab}, S_{mn}]$, is enough to process O_{ij} or longer than t_{ijk}.

Let S_k^d be the starting time of the dth idle time interval on M_k and C_k^d be the completion time. O_{ij} can be transferred to an earliest possible idle time interval of its machine which satisfies the following equation:

$$\max\{S_k^d, C_{ij-1}\} + t_{ijk} \leq C_k^d, (C_{ij} = 0, \text{if } j = 1) \quad (4)$$

After using the idle time interval, the starting time of O_{ij} is $\max\{S_k^d, C_{ij-1}\}$ and the idle interval is updated based on the starting and completion time of O_{ij}: (1) the idle time interval is removed; (2) the starting or completion time of the idle time interval is modified; (3) the idle time interval is replaced by two new shorter idle time intervals, like in Figure 2.

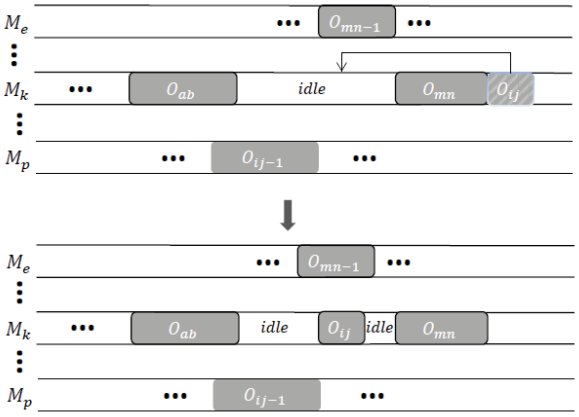

Fig. 2. First level local search

After decoding a chromosome, the operation sequence vector of the chromosome is updated according to new starting times of operations. The first level local search only finds for each operation the available idle time interval on its assigned machine. After generating the corresponding schedule with the first level search method, it is possible that there are still operations that can be allocated to available idle time intervals to benefit the fitness value. To achieve this, decoding the chromosome which has been updated with the first level local search is performed with the second level local search, and again operations are moved to available idle time intervals.

The second level local search: this search not only checks the idle time intervals on the assigned machine, but also the idle time intervals on alternative machines. An example of making use of the idle time interval on another machine is shown in Figure 3. Let S_{ijk} be the starting time and C_{ijk} be the completion time of O_{ij} on M_k. In this example, O_{ij} is assigned to M_k in the initial chromosome, we assume that O_{ij} can also be performed by M_e. Under the condition that the starting time of O_{ij} on M_k is later than the completion time of O_{ij-1}, the idle time intervals on all alternative machines which can process O_{ij} are checked. An idle time interval on M_e could be a choice and O_{ij} can be reallocated to M_e. In this example, the processing time of O_{ij} on M_e is even shorter then the processing time on M_k, therefore, this reallocation can at least benefit the total workload.

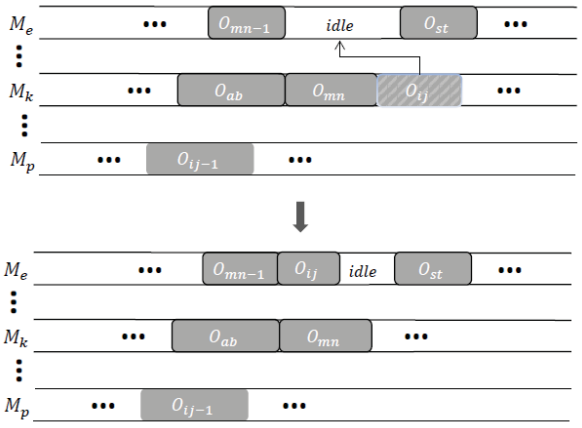

Fig. 3. Second level local search

With the second level local search, all available idle time intervals of an operation are checked one by one until the first "really" available idle time interval is found and then the operation is moved to that idle time interval. Any idle time interval on an alternative machine which can satisfy Equation 4 is an available idle time interval, while it must meet at least one of the following conditions to become a "really" available idle time interval.

1. The processing time of the operation on the new machine is shorter than that on the initially assigned machine if the available idle time interval is on a different machine;
2. The operation can be moved from the machine with the maximal makespan to another machine.
3. The operation can be moved from the machine with the maximal workload to another machine.

The total workload can be improved directly by the first condition; the motive of the second condition is to decrease the maximal makespan and the third condition can benefit the critical workload. After the reallocation of the operations with the second level local search, the corresponding schedule is obtained and objective values are calculated. While, instead of updating the chromosome immediately, the new objective values are compared with the previous objective values first, the chromosome can be updated only when at least one objective is better than its old value. This is to make sure that the new schedule is at least not worse than the old schedule (The new solution is not dominated by the old solution). Another difference between the first and second level local search is that the first level local search is performed once on every evaluation, and after the first level local search, the second level local search is performed once but only with a 30% probability to avoid local optima.

V. Experiments and Results

The experiments are implemented on the MOEA Framework (available from http://www.moeaframework.org). The

algorithms are tested on two sets of well-known FJSP benchmark instances: 4 Kacem instances (ka4x5, ka10x7, ka10x10, ka15x10) and 10 BRdata instances (Mk01-Mk10). Table III gives the scale of these instances. The first column is the name of each instance; the second column shows the size of the instance, in which n stands for the number of jobs and m the number of machines; the third column represents the number of operations; the fourth column lists the flexibility of each instance, which means the average number of alternative machines for each operation in the problem.

TABLE III
THE SCALE OF BENCHMARK INSTANCES

Instance	n $\rightarrow$ m	#Opr	Flex.
ka4x5	4 $\rightarrow$ 5	12	5
ka10x7	10 $\rightarrow$ 7	29	7
ka10x10	10 $\rightarrow$ 10	30	10
ka15x10	15 $\rightarrow$ 10	56	10
Mk01	10 $\rightarrow$ 6	55	2
Mk02	10 $\rightarrow$ 6	58	3.5
Mk03	15 $\rightarrow$ 8	150	3
Mk04	15 $\rightarrow$ 8	90	2
Mk05	15 $\rightarrow$ 4	106	1.5
Mk06	10 $\rightarrow$ 15	150	3
Mk07	20 $\rightarrow$ 5	100	3
Mk08	20 $\rightarrow$ 10	225	1.5
Mk09	20 $\rightarrow$ 10	240	3
Mk10	20 $\rightarrow$ 15	240	3

All experiments are performed with a population size of 100, each run of the algorithm stops based on a predefined number of evaluations, which is $10,000$ for Kacem instances and $150,000$ for BRdata instances. For each instance, the proposed algorithm is independently run 30 times. The resulting solution set of an instance is formed by merging all non-dominated solutions from 30 runs. The comparison between these sets is commonly adopted in the region of FJSP due to the difficulty of finding the complete non-dominated solutions.

The search space of the FJSP instances is complicated, therefore, the crossover probability is set to 1 and two random crossover operators are chosen each time (one for operation sequence and one for machine assignment). For Kacem instances, the mutation probabilities are set to 0.6. For BRdata instances, which include larger-scale and more complex problems, the MIP-EGO configurator is adopted to tune both insertion and swap mutation probabilities to find a proper setting for each problem because the parameters depend on the specific problem. The hypervolume of the solution set has been used in MIP-EGO as the objective value to tune mutation probabilities. Although the true PF for test instances are unknown, [7] provides the reference sets for Kacem and BRdata instances, which are formed by gathering all non-dominated solutions found by all implemented algorithms in [7] and also non-dominated solutions from other state-of-the-art MOFJSP algorithms. We define the reference point for calculating the hypervolume value based on the largest value in this reference set. To be specific, each objective value of the reference point is: $1.1 \times$ largest objective value of the respective dimension in the reference set. The origin is used as the ideal point. The other basic settings of MIP-EGO include using a 200 evaluation budget, random forest surrogate model, MIES as internal optimizer and ordinal search space.

Table IV shows the percentage of the evaluations which can achieve the largest hypervolume value (or the best PF) by MIP-EGO. It can be observed for Mk05 and Mk08 that all evaluations have obtained the largest hypervolume value. It means that all parameter values of mutation probabilities in MIP-EGO can achieve the best PF for them. We know that both problems have a low flexibility value which is the average number of alternative machines for each operation in the problem. On the contrary, for Mk06, Mk09 and Mk10, these problems have a large operation number and high flexibility. It seems that they are difficult to solve because there is only one best parameter setting for the mutation probabilities.

TABLE IV
PROBABILITY OF FINDING BEST CONFIGURATION

Mk01	Mk02	Mk03	Mk04	Mk05
73%	60%	95%	1%	100%
Mk06	Mk07	Mk08	Mk09	Mk10
0.5%	4.5%	100%	0.5%	0.5%

With the best parameter setting of the mutation probabilities for BRdata instances, we compared our experimental results with the reference set in [7]. Our algorithm can achieve the same Pareto optimal solutions as in the reference set for all BRdata instances except for Mk06, Mk09 and Mk10. At the same time, for Mk06 and Mk10, our algorithm can find new non-dominated solutions. Table V is the list of new non-dominated solutions obtained by our algorithm, each row of an instance is a solution with three objectives: makespan, total workload, and critical workload.

TABLE V
NEWLY ACHIEVED NON-DOMINATED SOLUTIONS

Mk06			Mk10		
61	427	53	218	1973	195
63	428	52	218	1991	194
63	435	51	219	1965	195
65	453	49	220	1984	191
66	451	49	225	1979	194
66	457	48	226	1954	196
			226	1974	194
			226	1979	192
			228	1973	194
			235	1938	199
			236	1978	193

Another comparison is between our algorithm (FJSP-MOEA) and MOGA [5], SEA [6] and MA1, MA2 [7]. In [7], there are several variants of the proposed algorithm with different strategies in the local search. We pick MA1 and MA2 as compared algorithms because they perform equally well or superior to other algorithms on almost all problems. Table VI displays the hypervolume values of the PF approximations from all algorithms and the new reference set which is formed by combining all solutions from the PF by all algorithms. The highest hypervolume value on each problem in all algorithms has been highlighted in bold. We observed that FJSP-MOEA and MA1, MA2 show the best and similar performance, and MOGA behaves the best for three of the BRdata instances. The

good performance of MOGA on three instances is interesting. MOGA has an entropy-based mechanism to maintain decision space diversity which might be beneficial for solving these problem instances. When using one best parameter setting, we also give the mean hypervolume and standard deviation from 30 runs on each problem in Table VII, the data shows the stable behaviour of our proposed FJSP-MOEA.

TABLE VI
HYPERVOLUME VALUES OF THE PF APPROXIMATIONS

Probs	MOGA	SEA	MA1	MA2	FJSP-MOEA	Ref
Mk01	0.00426	0.00508	**0.00512**	**0.00512**	**0.00512**	0.00512
Mk02	0.01261	0.01206	**0.01294**	**0.01294**	**0.01294**	0.01294
Mk03	**0.02460**	0.02165	0.02165	0.02165	0.02165	0.02809
Mk04	**0.06906**	0.06820	0.06901	0.06901	0.06901	0.07274
Mk05	0.00626	0.00635	**0.00655**	**0.00655**	**0.00655**	0.00655
Mk06	0.05841	0.06173	0.06585	0.06692	**0.06709**	0.07065
Mk07	0.02244	0.02132	**0.02269**	**0.02269**	**0.02269**	0.02288
Mk08	**0.00418**	0.00356	0.00361	0.00361	0.00361	0.00428
Mk09	0.01547	0.01755	0.01788	**0.01789**	0.01785	0.01789
Mk10	0.01637	0.01778	0.02145	**0.02196**	0.02081	0.02249

TABLE VII
MEAN HYPERVOLUME AND STD WITH THE BEST PARAMETER SETTING

Problem	Mk01	Mk02	Mk03	Mk04	Mk05
Mean HV	0.0050	0.0122	0.0216	0.0672	0.0064
Std	0	0.0003	0.0001	0.0004	0.0001
Problem	Mk06	Mk07	Mk08	Mk09	Mk10
Mean HV	0.0598	0.0222	0.0036	0.0174	0.0186
Std	0.0019	0.0003	0	0.0002	0.0006

For Kacem instances and with fixed mutation probabilities, our obtained non-dominated solutions are the same as the PF in the reference set. MA1 and MA2 also achieved the best PF for all Kacem instances, but our algorithm uses far less computational resources. The proposed FJSP-MOEA uses only a population size of 100 whereas the population size of MA algorithms is 300. FJSP-MOEA uses only $10,000$ objective function evaluations, whereas MA uses $150,000$ evaluations. In terms of computational resources, the proposed FJSP-MOEA can therefore be used on smaller computer systems, entailing broader applicability, and possibly also in real-time algorithm implementations such as dynamic optimization.

VI. CONCLUSIONS

A novel multi-objective evolutionary algorithm for the MOFJSP is proposed. It uses multiple initialization approaches to enrich the first-generation population, and various crossover operators to create better diversity for offspring. The reason of using multiple approaches and operators is not only because any one of them cannot achieve good performance, also to simplify the algorithm procedure. To determine the optimal mutation probabilities, the MIP-EGO configurator is adopted to replace a sensitivity analysis to automatically generate proper parameter settings. Besides, local search is employed with different neighborhood sizes levels to aid more accurate convergence. The proposed customization approach in principle can be combined with almost all MOEAs. In this paper, we incorporate it with one state-of-the-art MOEA, namely NSGA-III, to solve the MOFJSP, and the new algorithm can find all Pareto optimal solutions in literature for most problems, and even new Pareto optimal solutions for the large scale instances.

In this paper, we show the ability of MIP-EGO in finding the optimal mutation probabilities. There is more potential of the automated parameter configuration domain to benefit EA, for example, tuning the initialization and crossover configuration, the population size, and so on; also, other configurators can be involved. However, so far the efficiency of the existing tuning framework is limited when it comes to a larger number of parameters. It would therefore be a good topic of future research to find more efficient implementations of these.

ACKNOWLEDGMENT

This work is part of the research programme Smart Industry SI2016 with project name CIMPLO and project number 15465, which is (partly) financed by the Netherlands Organisation for Scientific Research (NWO).

REFERENCES

[1] Garey, M.R., Johnson, D.S. and Sethi, R., 1976. The complexity of flowshop and jobshop scheduling. Mathematics of operations research, 1(2), pp.117-129.
[2] van Stein, B., Wang, H. and Bäck, T., 2018. Automatic configuration of deep neural networks with EGO. arXiv preprint arXiv:1810.05526.
[3] Deb, K. and Jain, H., 2013. An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, part I: solving problems with box constraints. IEEE transactions on evolutionary computation, 18(4), pp.577-601.
[4] Chaudhry, I.A. and Khan, A.A., 2016. A research survey: review of flexible job shop scheduling techniques. International Transactions in Operational Research, 23(3), pp.551-591.
[5] Wang, X., Gao, L., Zhang, C. and Shao, X., 2010. A multi-objective genetic algorithm based on immune and entropy principle for flexible job-shop scheduling problem. The International Journal of Advanced Manufacturing Technology, 51(5-8), pp.757-767.
[6] Chiang, T.C. and Lin, H.J., 2013. A simple and effective evolutionary algorithm for multiobjective flexible job shop scheduling. International Journal of Production Economics, 141(1), pp.87-98.
[7] Yuan, Y. and Xu, H., 2013. Multiobjective flexible job shop scheduling using memetic algorithms. IEEE Transactions on Automation Science and Engineering, 12(1), pp.336-353.
[8] Das, I. and Dennis, J.E., 1998. Normal-boundary intersection: A new method for generating the Pareto surface in nonlinear multicriteria optimization problems. SIAM journal on optimization, 8(3), pp.631-657.
[9] Zhang, G., Gao, L. and Shi, Y., 2011. An effective genetic algorithm for the flexible job-shop scheduling problem. Expert Systems with Applications, 38(4), pp.3563-3573.
[10] Kacem, I., Hammadi, S. and Borne, P., 2002. Approach by localization and multiobjective evolutionary optimization for flexible job-shop scheduling problems. IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews), 32(1), pp.1-13.
[11] Pezzella, F., Morganti, G. and Ciaschetti, G., 2008. A genetic algorithm for the flexible job-shop scheduling problem. Computers & Operations Research, 35(10), pp.3202-3212.
[12] Xing, L.N., Chen, Y.W. and Yang, K.W., 2009. An efficient search method for multi-objective flexible job shop scheduling problems. Journal of Intelligent Manufacturing, 20(3), pp.283-293.
[13] Glover, F.W. and Kochenberger, G.A. eds., 2006. Handbook of metaheuristics (Vol. 57). Springer Science & Business Media.
[14] Zhang, C.Y., Rao, Y.Q., Li, P.G. and Shao, X.Y., 2007. Bilevel genetic algorithm for the flexible job-shop scheduling problem. Chinese Journal of Mechanical Engineering, 43(4), pp.119-124.
[15] Zhang, C., Li, P., Rao, Y. and Li, S., 2005, March. A new hybrid GA/SA algorithm for the job shop scheduling problem. In European Conference on Evolutionary Computation in Combinatorial Optimization (pp. 246-259). Springer, Berlin, Heidelberg.
[16] Teekeng, W. and Thammano, A., 2012. Modified genetic algorithm for flexible job-shop scheduling problems. Procedia Computer Science, 12, pp.122-128.