# Genetic Algorithm for FJSP Problem

WeiLi Zhong
School of Peking University
Email: qq994478@gmail.com

***Abstract*—In this paper a genetic algorithm (GA) is developed to create a feasible and active schedule for the flexible job-shop scheduling problems with the aims of minimizing completion time of all jobs, i.e. makespan. In the proposed algorithm, an enhanced solution coding is used. To generate high quality initial populations, we designed an Operation order-based Global Selection (OGS), which is taken into account both the operation processing times and workload of machines while is assigning a machine to the operation which already is ordered randomly in chromosome operation sequence part. The precedence operation crossover (POX) and job-based crossover (JBX) are used appropriately and furthermore an intelligent mutation operator is carried out. The proposed algorithm is applied on the benchmark data set taken from literature. The results demonstrated efficiency and effectiveness of the algorithm for solving the flexible job shop scheduling problems.**

## I. Introduction

The scheduling of flexible job-shop can be described as a set of independent jobs, $J = \{j_1, j_2, ..., j_n\}$, each of which consists of a sequence of operations, $O_{j,1}$,$O_{j,2}$, ..., $O_{j,h}$. The operations of each job should be performed in a predetermined sequence on a subset of $M_{i,j} \subseteq M$ capable machines from a given set of $M = \{m_1, m_2, ..., m_k\}$ machines. The problem is partial flexible, which means, a proper subset of machines can perform operation $O_{j,h}$. Processing time of operations consists of all preparation and movement times with no sequence dependent setup times. All machines are available at time 0, and they process one job at a time. All materials are available at time 0. Each operations of the job once started must be completed without any interruption, i.e. no preemption.

The flexible job-shop scheduling problem, which this paper attempts to solve is that, at first, order the operations of jobs (scheduling problem) and then allocates them to the suitable machines (routing problem) with the aims of minimizing makespan, i.e. the total time that is required for the completion of all jobs. The makespan can be defined $\{C_j\}$, where $C_j$ is the completion time of job $j$.

For the simplicity of presenting the algorithm through this paper, an instance of flexible job shop with 3 jobs consisting of 9 operations and 5 machines is designed, which is shown in Table 1.

The first Column in Table 1 corresponds to jobs. The second column corresponds to the operations. Each cell in columns 3 to 7 denotes the processing times of operations on the corresponding machines. The symbol "X" in the table means that a machine cannot perform the corresponding operation.

The remainder of this paper is properly organized as follows. Section 2 explains the structure and mechanism of the proposed GA. Section 3 presents computational results. Finally, the conclusion and future researches are provided in section 4.

## II. Genetic Algorithm for Flexible Job Shop Scheduling Problem

Genetic algorithm (GA) is a random search optimization technique, which is based on the principles of genetic and natural selection. This technique was inspired by Charles Darwin's theory of evolution and was initially introduced by John Holland in the 1970s [1]. Based on the mechanism of GA, first, the problem is encoded into a set of strings, i.e. chromosome, which consists of several bits. Then it operates on the strings to simulate the process of evolution [2]. The full of proposed GA is shown in Fig. 1.

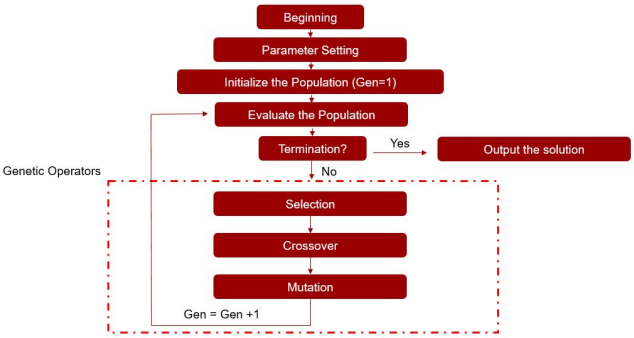


Fig. 1. The proposed GA

### A. Chromosome Representation

TABLE I
OPERATIONS PROCESSING TIME

| Job | Operation | M1 | M2 | M3 | M4 | M5 |
|---|---|---|---|---|---|---|
| J1 | $O_{11}$ | X | 5 | 7 | X | 9 |
| | $O_{12}$ | 8 | X | 9 | 9 | 12 |
| J2 | $O_{21}$ | 3 | X | 5 | 2 | X |
| | $O_{22}$ | X | 5 | 8 | 7 | X |
| | $O_{23}$ | 9 | 7 | X | 6 | 10 |
| J3 | $O_{31}$ | 4 | 3 | 5 | X | 3 |
| | $O_{32}$ | X | 6 | 8 | X | 7 |
| | $O_{33}$ | 12 | 10 | X | 10 | 13 |
| | $O_{34}$ | X | 4 | 6 | X | 5 |

Each solution consists of two vectors: operation sequencing vector and machine assignment vector. The length of each vector is equal to the total number of operations of all jobs. The numbers in the operation sequencing vector are job indices. Each job $j$ appears in the operation sequence vector exactly $s_j$ times to represent its $s_j$ ordered operations. In addition, the numbers in the machine assignment vector are machine indices. Fig.2 shows an example of chromosome representation for the problem instance in Table 1.

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 3 | 2 | 3 | 1 | 3 | 3 | 2 | 2 | 1 |
| 5 | 1 | 3 | 3 | 4 | 5 | 4 | 2 | 1 |

Fig. 2. Chromosome representation for Table 1

This chromosome can be interpreted as shown in Fig. 3. For example, the first number in the operation sequence vector represents the first operation of job 3, $O_{31}$, which should be performed on machine 5, $M_5$, in the machine assignment vector. The second number in the operation sequence vector represents the first operation of job 2, $O_{21}$, which should be performed on machine 1, $M_1$, and so forth.

| Priority Number | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|
| | $O_{31}$ | $O_{21}$ | $O_{32}$ | $O_{11}$ | $O_{33}$ | $O_{34}$ | $O_{22}$ | $O_{23}$ | $O_{12}$ |
| Operation Sequencing Vector | 3 | 2 | 3 | 1 | 3 | 3 | 2 | 2 | 1 |
| Machine Assignment Vector | 5 | 1 | 3 | 3 | 4 | 5 | 4 | 2 | 1 |
| | $M_5$ | $M_1$ | $M_3$ | $M_3$ | $M_4$ | $M_5$ | $M_4$ | $M_2$ | $M_1$ |

Fig. 3. Chromosome interpretation

### *B. Decoding the Chromosome into a Feasible and Active Schedule*

In this research, an active schedule is considered during decoding approach. Active schedule is a feasible schedule, in which none of the operations could be started earlier without delaying some other operations or changing an order constraint[4].

### *C. Initial Population*

Operations of jobs are randomly sequenced in the operation sequencing vector in order to have diversity in the populations. However, two methods are used for assigning machines to the operations as follows:

The first method for assigning a machine is Random Selection. The second method for machine assignment is Operation order-based Global Selection (OGS), in which we developed an "approach by localization" proposed by [3] and global selection designed by [4] in order to generate higher quality initial populations. This method has good exploration in the search space and considers both the operation processing times and workload of machines while is assigning a machine to the operation which already is ordered randomly in chromosome 'operation sequence part. The detailed steps of OGS are as follows:

Step 1: Create an array to record selected machine and initialize each elements to 0;

Step 2: Make a copy from the processing time table in Table 1 (called P) and save it as update processing time table (called P1).

Step 3: Sort all operations in operation sequence vector and record the indices of all operations;

Step 4: Select the first operation according to the recorded indices;

Step 5: Select a machine with the shortest processing time from the capable machine in P1 and assign it to the operation. If the processing times of machines are the same, one of them is selected randomly;

Step 6: Record the selected machine in the machine's array according to selected machine in P1;

Step 7: Update the processing time of column in P1, in which selected machine belongs to it, i.e. add the selected machine's processing time from P to the processing times of column under selected machine in P1, except processing time of selected machine.

Step 8: Select the next operation and execute step 5 and 7 until all operations are selected;

Suppose operations are sequenced as shown in Fig.2 then OGS is implemented as illustrated in Fig.4.

### *D. Genetic Algorithm Operators*

In order to obtain a better chromosome, operators of GA such as selection, crossover, and mutation are carried out in each generation as follows.

*1) Selection Operator:* In GA, the selection operator is used to select the individuals according to the fitness. In this paper, two selection operators are adopted. The first one is the elitist selection scheme. This method reproduces $P_r$ × Popsize ($P_r$ is the reproduction probability; Popsize is the size of the population) individuals with good fitness in parents to the offspring. The other one is the tournament selection scheme. In tournament selection, a number of individuals are selected at randomly (dependent on the tournament size b, typically between 2 and 7, in this paper b=2) from the population and the individual with better fitness is selected. The tournament selection approach allows a tradeoff to be made between exploration and exploitation of the gene pool. This scheme can modify the selection pressure by changing the tournament size.

*2) Crossover Operators of Sequencing:* In this paper, two crossover operators have been adopted for the OS string. In the procedure of GA, one crossover operator is selected randomly (50%) to crossover the OS string. The first one is the precedence operation crossover (POX). The basic working procedure of POX is described as follows (two parents are denoted as P1 and P2; two offspring are denoted as O1 and O2):

Step 1: The Job set $J = \{J_1, J_2, ..., J_n\}$ is divided into two groups Jobset1 and Jobset2 randomly;

Step 2: Any element in P1 which belongs to Jobset1 are appended to the same position in O1 and deleted in P1; any

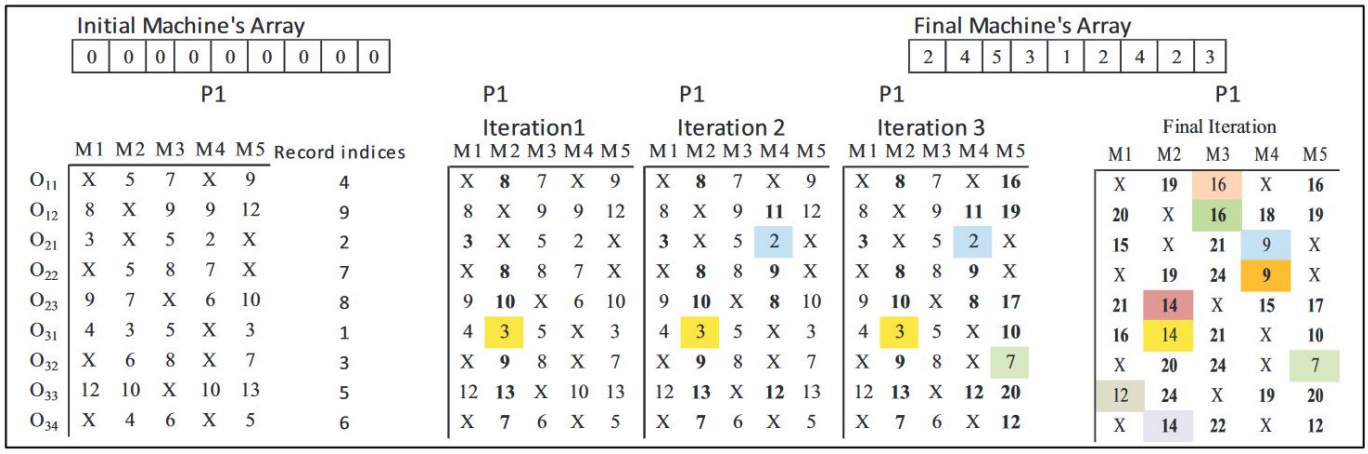


Fig. 4. Operation order-based global selection method for machine assignment

element in P2 which belongs to Jobset1 are appended to the same position in O2 and deleted in P2;

Step 3: the remaining elements in P2 are appended to the remaining empty positions in O1 seriatim; and the remaining elements in P1 are appended to the remaining empty positions in O2 seriatim.

The second crossover operator for OS string is the job-based crossover (JBX). The basic working procedure of JBX is described as follows (two parents are denoted as P1 and P2; two offspring are denoted as O1 and O2):

Step 1: The Job set $J = \{J_1, J_2, ..., J_n\}$ is divided into two groups Jobset1 and Jobset2 randomly;

Step 2: Any element in P1 which belongs to Jobset1 are appended to the same position in O1 and deleted in P1; any element in P2 which belongs to Jobset2 are appended to the same position in O2 and deleted in P2;

Step 3: The remaining elements in P2 are appended to the remaining empty positions in O1 seriatim; and the remaining elements in P1 are appended to the remaining empty positions in O2 seriatim.

For the OS string, Fig. 5(a) shows an example of the POX crossover operator and the Fig. 5(b) describes an example of the JBX crossover operator.

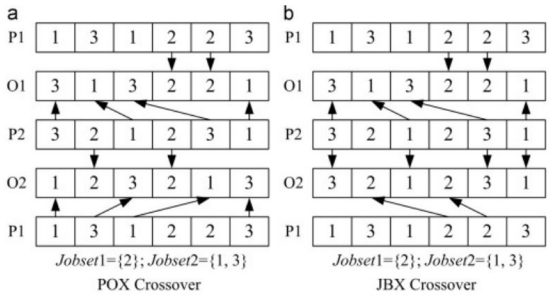


Fig. 5. Crossover operators for OS string

*3) Crossover Operators of Assignment:* For the MS string, a two-point crossover has been adopted here as the crossover operator. In this operator, two positions are selected randomly at first. Then two offspring strings are generated by swapping all elements between the positions of the two parents strings. Because every gene in this string denotes the selected machine for the fixed operations, the operation number does not change in the whole searching process. This crossover can ensure to generate feasible offspring if the selected parents are feasible. Fig. 6 shows an example of the crossover operator for MS string. Firstly, selecting two feasible parents (P1 and P2); then, selecting two positions randomly, in this example, the 2nd and 5th positions are selected; adopting the elements of P1 before the 2nd position and after the 5th position to the same positions in O1, and adopting the elements of P2 from the 3rd position to the 4th position to the same positions in O1, using the same process to generate the O2. Base on the whole process, we can see that if two parents are feasible, the offspring will be feasible.

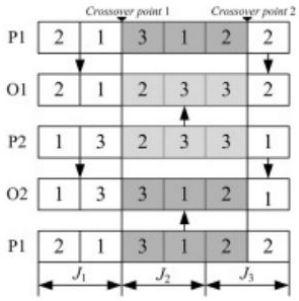


Fig. 6. Crossover operator for MS string

*4) Mutation Operator of Sequencing:* In this paper, two mutation operators have been adopted for the OS string. In the procedure of GA, one mutation operator is selected randomly (50%) to mutate the OS string. The first one is the swapping mutation. The basic working procedure of swapping mutation is described as follows (one parent is denoted as P; one offspring is denoted as O):

Step 1: Select two positions in the P;

Step 2: Swap the elements in the selected positions to generate the O;

The second mutation operator for OS string is the neighborhood mutation method. The basic working procedure of this method is described as follows (one parent is denoted as P; one offspring is denoted as O):

Step 1: Select 3 elements in the P (the values of these elements are different), and generate all the neighborhood OS strings;

Step 2: Choose the one in the neighborhood OS strings randomly and set it as the current OS string, it is the O.

For the OS string, Fig. 7(a) shows an example of the swapping mutation operator and the Fig. 7(b) describes an example of the neighborhood mutation operator. In Fig. 7(a), the 2nd and 5th positions are selected. The offspring is generated by swapping the elements in these positions. In Fig. 7(b), the 1st, 4th and 6th positions are selected. And then, generating 5 neighborhood OS strings, one of them is selected randomly as the offspring.

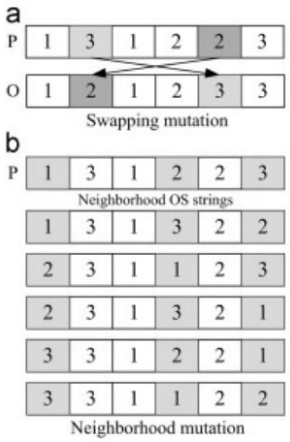


Fig. 7. Mutation operators for OS string

*5) Mutation Operator of Assignment:* For the MS string, a mutation operator is designed as follows:

Step 1: Select r positions in the parent (r is the half of the length of the MS string);

Step 2: For each position (according to one operation), change the value of this selected position to the other machine in the machine set of the corresponding operation.

Fig. 8 shows an example of the mutation operator for MS string. In this example, r equals to 3 (the length is 6). And the 1st, 4th and 6th positions are selected. They represent the selected machine for $O_{11}$, $O_{21}$ and $O_{32}$ respectively. The offspring is generated by changing the value of these positions to the other machine in the machine set of the corresponding operation.

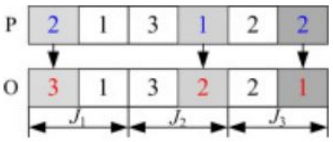


Fig. 8. Mutation operator for MS string

### *E. Fitness Evaluation Function*

Chromosomes, which represent the solutions, are evaluated based on the completion time of all jobs. Since the objective of this paper is to minimize makespan, the solutions with the lower completion time are preferable and are recorded.

### *F. Stopping Criterion*

The stopping criterion is set based on the maximum number of iterations and when it is reached the best schedule is given as output.

## III. COMPUTATIONAL RESULTS

The proposed GA was coded in Python and executed on a personal computer. The processor used was an Intel Core TM with a speed of 2.20 GHz and 16GB RAM. During several experiments, different values of parameters were tested. Computational results proved that the following values for the parameters of the proposed GA are more effective. The population size, 400; number of generations, 200; crossover probability, 0.8, and mutation probability, 0.1.

To show the performance of proposed algorithm, a so-called benchmark FJSP data set (Brandimarte data) has been taken from [5]. This data set consists of ten problems MK01-MK010, in which the number of jobs is in the range of 10 to 20, number of machines 4 to 15, and number of operations for each job ranges from 5 to 15. The proposed GA is applied on these ten problems, and our results are compared with results from recent literatures [6][7][8] as shown in Table 2.

In Table 2, the first column shows the problem name. The second column, n*m, reports the size of the problem, where n and m denote the jobs and machines, respectively. The third column reports the total number of all operations of all jobs. Fourth and fifth columns report the best lower bound and upper bound found to date, respectively. The best makespan obtained by our GA over 10 runs is reported in column sixth. The best results of three algorithms to be compared with the results of our GA are reported in columns 7 to 9. The results in Table 2 show the performance and effectiveness of our GA for solving the FJSPs. Fig. 9 illustrates the Gantt chart of the problem MK02.

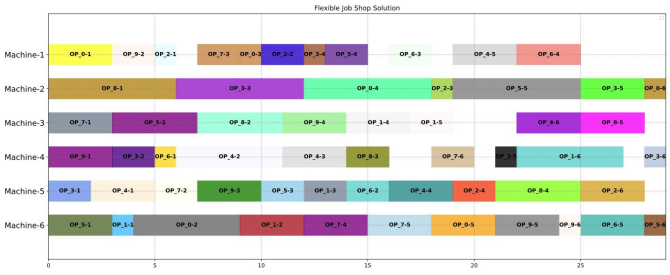


Fig. 9. Gantt chart of problem MK02

TABLE II
COMPARISON WITH OTHER ALGORITHM ON 10 FJSP INSTANCES FROM BRANDIMARTE

| Problem | n*m | LB | UB | Pro-posed GA | GA Pezella et al. | AIA Bagheri et al. | eGA Zhang et al. |
|---|---|---|---|---|---|---|---|
| MK01 | 10*6 | 36 | 42 | 40 | 40 | 40 | 40 |
| MK02 | 10*6 | 24 | 32 | 28 | 26 | 26 | 26 |
| MK03 | 15*8 | 150 | 204 | 211 | 204 | 204 | 204 |
| MK04 | 15*8 | 48 | 81 | 62 | 60 | 60 | 60 |
| MK05 | 15*4 | 168 | 186 | 173 | 173 | 173 | 173 |
| MK06 | 10*15 | 33 | 86 | 62 | 63 | 63 | 58 |
| MK07 | 20*5 | 133 | 157 | 140 | 139 | 140 | 144 |
| MK08 | 20*10 | 523 | 523 | 523 | 523 | 523 | 523 |
| MK09 | 20*10 | 299 | 369 | 307 | 311 | 312 | 307 |
| MK10 | 20*15 | 165 | 296 | 215 | 212 | 214 | 198 |

## IV. CONCLUSION

In this paper, a genetic algorithm has been proposed for solving the flexible job-shop scheduling problems with the aims of minimizing the completion time of all jobs. To achieve high quality solutions, we designed OGS and intelligent mutation, which our results demonstrated the effectiveness of proposed GA for solving FJSPs. In the future, it would be interesting to investigate other objectives like minimizing tardiness, lateness, machine workload, and total machine workload.

## REFERENCES


[1] Sels, V., Steen, F., and Vanhoucke, M. (2011). Applying a hybrid job shop procedure to a Belgian manufacturing company producing industrial wheels and castors in rubber. Computers and Industrial Engineering, 61(3), 697-708.

[2] Zhou, H., Feng, Y., and Han, L. (2001). The hybrid heuristic genetic algorithm for job shop scheduling. Computers and Industrial Engineering, 40(3), 191-200.

[3] Rahnamayan, S., Tizhoosh, H. R., and Salama, M. M. A. (2007). A novel population initialization method for accelerating evolutionary algorithms. Computers and Mathematics with Applications, 53(10), 1605-1614.

[4] Paulli, J. (1995). A hierarchical approach for the FMS scheduling problem. European Journal of Operational Research, 86(1), 32–42.

[5] Brandimarte, P. (1993). Routing and scheduling in a flexible job shop by taboo search. Annals of Operations Research, 41, 157–183.

[6] Bagheri, A., Zandieh, M., Mahdavia, I., and Yazdani, M. (2010). An artificial immune algorithm for the flexible job-shop scheduling problem, Journal of Future Generation Computer Systems: pp. 533-541.

[7] Pezzella, F., Morganti, G., and Ciaschetti, G. (2008). A genetic algorithm for the Flexible Job-shop Scheduling Problem. Computers and Operations Research, 35(10), 3202-3212.

[8] 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), 3563-3573.