An effective Genetic Algorithm for the Flexible Job Shop Scheduling Problems

IMEN DRISS¹, KINZA NADIA MOUSS², ASSIA LAGGOUN³

AUTOMATICS & MANUFACTURING ENGINEERING LABORATORY
University of Batna, Algeria.
Idrissamina@hotmail.fr
kinzmouss@yahoo.fr
a.laggoun12@yahoo.fr

Abstract

This present paper will deal with a flexible job shop scheduling problem in order to minimize the makespan. We will propose actually a Genetic algorithm method. In that effective algorithm: a new chromosome representation will be used to conveniently represent a solution for the FJSP, an advanced crossover and mutation operators will be suggested to adapt to chromosome structures. The performance of our proposed approach will be evaluated by a lot of benchmark instances taken from literature. The experimental results will show that the proposed algorithm is a feasible and effective approach for the flexible job shop scheduling problem.

Keywords - FJSP; Scheduling ; Genetic Algorithm ; Chromosome representation.

1. Introduction

Job-shop scheduling problem (JSP) is a branch of production scheduling and combinatorial optimization problems. The classical JSP consists of scheduling a set of jobs on a set of machines under the constraint; each job has a specified processing order. The flexible job-shop problem (FJSP) is an extension of the classical job shop problem in which each operation must be processed on a machine chosen among a set of available ones.

The problem of scheduling job in FJSP can be composed of sub-problems: assigning the operation to machine (the routing problem) and sequencing the operations on the machines (the sequencing problem) in order to minimize the performance indicators. Therefore, the combination of two decisions presents additional complexity. Thus, the FJSP problem is an NP-hard one since it is an extension of the job shop scheduling problem (JSP) is proven to be NP-hard[1].

Last years, meta-heuristics were used to solve FJSP namely, tabu search, simulated annealing, genetic algorithm and particle swarm optimization.

In this research, we suggest a new genetic algorithm in order to solve flexible job shop scheduling problems to minimize the Makespan. We create a new chromosome representation called ‘permutation job’. In fact, this method leads us to find a new coding scheme of the individuals that respects all constraints of the FJSP. At the same time, we employ different strategies for crossover and mutation operators. Computational results show that the proposed algorithm could provide good results.

The paper is organized as follows: the problem definition and formulation are presented in section 2. Literature review is mentioned in section 3. The effective genetic algorithm (eGA) is expanded in section 4. Computational results are presented in section 5. Conclusion and future work are dealt within section 6.

2. Definition and formalization problem

2.1 Problem Description

We focus on flexible job-shop scheduling problems composed of the following elements:

- Jobs. \( J = \{ J_1, \ldots, J_n \} \) is a set of \( n \) jobs to be scheduled. Each job \( J_i \) consists of set of predetermined operations. \( O_{ij} \) is the operation of job \( J_i \). All jobs are released at time 0.

- Machines. \( M = \{ M_1, \ldots, M_m \} \) is a set of \( m \) machines. Each machine can process only one operation at a given time; and each operation can be processed without interruption. All machines are available at time 0.

- Flexibility. The FJSP is classified in two types as follows: [2]
  - Total FJSP (T-FJSP): each operation can be processed on any machine of \( M \) existing machines in the shop floor.
Partial FJSP (P-FJSP): each operation can be processed on one machine of a subset of M existing machines in the shop floor.

- Constraints: Are rules that limit the possible assignments of the operations. They can be divided mainly into the following situations:
  - Each operation can be processed by only one machine at a time (disjunctive constraint).
  - Each operation, which has started, runs to completion (non-preemption condition).
  - Each machine performs operations one after another (capacity constraint).
  - Although there are no precedence constraints among operations of different jobs, the predetermined sequence of operations for each job forces each operation to be scheduled after all predecessor operations (precedence conjunctive constraint).
  - The machine constraints emphasize the operations can be processed only by the machine from the given set (resource constraint).

The hypotheses considered in this paper are summarized as follows:
- Each operation can be processed without interruption on one of a set of available machines;
- Jobs are independent and no priorities are assigned to any job type;
- All jobs are released at time 0, and all machines are available at time 0 too;
- Breakdowns are not considered;
- The order of operations for each job is predefined and cannot be modified.

Objective: Is to find a schedule that has minimum time required to complete all operations (minimum Makespan).

In order to simplify the presentation of the algorithm; we designed a sample instance of FJSP which will be used a long this paper. Table 1 gives the dataset of a P-FJSP including 2 jobs operated on 4 machines where rows correspond to operations and columns correspond to machines. Each cell denotes the processing time of that operation on the corresponding machine. We can represent this system as a graph figure 1.

### Table 1: Processing time table of an instance of P-FJSP

<table>
<thead>
<tr>
<th>Job</th>
<th>Operations</th>
<th>M1</th>
<th>M2</th>
<th>M3</th>
<th>M4</th>
</tr>
</thead>
<tbody>
<tr>
<td>J1</td>
<td>O1i</td>
<td>5</td>
<td>2</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>O12</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>O13</td>
<td></td>
<td>-</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>J2</td>
<td>O21</td>
<td>4</td>
<td>5</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>O22</td>
<td>3</td>
<td>2</td>
<td>-</td>
<td>1</td>
</tr>
</tbody>
</table>

## 2.2 Problem formulation

Some symbols used in our work are listed as follows.

- n: Number of the total jobs.
- m: Number of the total machines.
- j: Index of jth job.
- i: Index of ith operation of job j.
- $I_{io}$: Number of the total operation of job j.
- $O_{ij}$: jth Operation of job j.
- $p_{ijk}$: Processing time of operation $O_{ij}$ on machine k.
- $s_{ij}$: Start time of operation $O_{ij}$.
- $e_{ijk}$: End time of operation $O_{ij}$ on machine k.
- $L=\sum_{i}I_{io}$: The sum of all operation of all jobs.
- H: Very large positive integer.

### Objective function:

Minimize $\max\left(C_{ij_1}, C_{ij_2}, \ldots, C_{ij_n}\right)$ (1)

### Subject to:

\begin{align}
C_{ij} - S_{ij} - \sum_{k \neq i} p_{ijk} X_{ijk} &= 0 \quad \forall j, i \quad (2) \\
C_{ij} - C_{ij} + H(1 - Y_{ij}) + H(1 - X_{ij}) + (1 - X_{ij}) &\geq p_{ijk} \quad (3) \\
C_{ij} - C_{ij} + H(Y_{ij}) + H(1 - X_{ij}) + (1 - X_{ij}) &\geq p_{ijk} \quad (4) \\
S_{ij} &\geq 0 \quad \forall i, j \quad (5) \\
S_{ij+1} - c_{ij} &\geq 0 \quad \forall i, j \quad j = 1, \ldots, l - 1 \quad (6) \\
\sum_{k \neq i} \sum_{k \neq i} X_{ijk} &= 1 \quad \forall i, j \quad (7) \\
X_{ijk} &= \begin{cases} 1 & \text{if operation } O_{ij} \text{ is assigned to machine } k \\ 0 & \text{otherwise} \end{cases} \quad (8) \\
Y_{ij} &= \begin{cases} 1 & \text{if operation } O_{ij} \text{ proceeds on machine } k \\ 0 & \text{otherwise} \end{cases} \quad (9)
\end{align}

Objective (1) minimizes. The constraint set (2) imposes that the difference between the completion time and the starting time of an operation is equal to its processing time on the machine assigned to it. Constraint set (3) and (4) ensure that no two operations can be processed simultaneously on the same machine. This disjunctive constraint (3) becomes inactive when $Y_{ij} = 0$ and the disjunctive constraint (4) becomes inactive when $Y_{ij} = 1$. Constraint set (5) ensures that the start time of an operation is always positive. Constraint set (6) represents the precedence relationship among various operations of a job. Constraint set (7) imposes that an operation can only be assigned to one machine.[16]

### 3. Literature review

Bruker and Schlie were the first to consider this problem. They developed a polynomial algorithm for solving the flexible job shop scheduling problem with two jobs. However, exact algorithms are not effective for solving FJSP and large instances.
Brandimarte was the first to apply the decomposition approach into the FJSP [4]. He solved the routing sub-problem by using some existing dispatching rules and then focused on the scheduling sub-problem, solved by using a tabu search heuristic. Paulli applied hierarchical approach results, but it is rather difficult to be implemented in real operations. [5] and Mesghouni et al were the first to model GA known as parallel job representation for solving FJSP [6]. Chen et al proposed a GA that uses an A-B string representation to solve FJSP for minimum Makespan time criterion [7]. Kacem, Hammad, and Borne proposed a genetic algorithm controlled by the assignment model generated by the approach of localization (AL) to mono-objective and multi-objective FJSP. [8] Ho and Tay proposed a GA based tool, namely GENACE, for solving the FJSP for minimum Makespan time criterion. The chromosome representation consists of two components, one component for machine selection and the other for operation sequence. Their methodology first generates an initial population using composite dispatching rules. A cultural evolution is then applied to preserve knowledge of schemata and resource allocations learned over each generation. The knowledge or belief spaces in turn influence mutation and selection of individuals. [9] Zhang and Gen proposed a method called multistage based genetic algorithm to solve FJSP problem [10]. Mehrabadi and Fattahi presented a mathematical model and tabu search algorithm to solve the flexible job shop scheduling problem with sequence dependent on setups for minimizing the Makespan[11]. Ho et al proposed an architecture for learning and evolving of flexible job shop schedules for minimum Makespan criterion called learnable genetic architecture (LEGA), a generalization of their previous approach GENACE (Ho and Tay 2004) The population generator module generates a set of feasible schedules equal to the population size using composite dispatching rules and then encodes it into chromosomes of initial population for subsequent evolution in the EA module. During genetic evolution, the SL module modifies the offspring schedules to improve solution quality and to preserve feasibility based on a memory of conserved schemas resolved from sampled schedules sent dynamically from EA module [12]. Gao and Gen developed a hybrid optimization strategy for FJSP multi objective (min Makespan, min maximal machine workload and min total workload) by combining the genetic algorithm and the bottleneck shifting [13]. Tay and Ho proposed a genetic programming (GP) based approach for evolving effective composite dispatching rules for solving the multi-objective FJSP [14]. Girish and Jawahar (2008) proposed a GA for the FJSP for minimum makespan time criterion. The chromosome representation of their proposed GA consists of two strings: one string for machine assignment and the other string for sequencing the operations on the assigned machines using Giffler and Thompson schedule generation procedure (Giffler and Thompson 1960). The chromosomal representation of their proposed GA does not require a repair mechanism and is capable to rummage through the entire search space. [15]. Ponnambalam et al Genetic algorithm (GA) based heuristics that have adopted Giffler and Thompson (GT) procedure, an efficient active feasible Schedule for Makespan time criterion [16]. Giovanni and Pezzella proposed an Improved Genetic Algorithm to solve the Distributed and Flexible Job-shop Scheduling problem to minimize the Makespan [17]. Sun et al presented a research on flexible job shop scheduling problem based on a modified GA [18]. Motaghed et al presented an effective hybrid genetic algorithm to solve the multi-objective flexible job shop scheduling problems [19]. Guohui et al proposed an effective genetic algorithm for solving the flexible job-shop scheduling problem (FJSP) to minimize Makespan criteria[20] Zhang et al proposed a genetic algorithm with tabu search procedure for Flexible Job Shop Scheduling Problem (FJSP) with transportation constraints and bounded processing times to minimize the Makespan and the storage of solutions [21]. Chen et al proposed Genetic Algorithm (GA) and Grouping Genetic Algorithm (GGA) for job shop scheduling problem with parallel machines and reentrant process [22].

4. An effective Genetic Algorithm for flexible job shop scheduling problem

4.1 Basis of genetic Algorithm

Genetic algorithms (GAs) are search methods based on principles of natural selection and genetics [23-24]. The interest in heuristic search algorithms with underpinnings in natural and physical processes began as early as the 1970s, when Holland first proposed genetic algorithms [25]. The fundamental underlying mechanism to start the search GAs is initialized with a population of individuals. The individuals are encoded as chromosomes in the search space. GAs use mainly two operators namely, crossover and mutation to direct the population to the global optimum. Crossover allows exchanging information between different solutions (chromosomes) and mutation increases the variety in the population. After the selection and evaluation of the initial population, chromosomes are selected the crossover and mutation operators are applied. Then the new population is formed. This process is continued until a termination criterion is met. This process is continued until a termination criterion is met[26-27].

![Figure.2 .The cycle GAs](image)

4.2 Chromosome representation

Our chromosome representation has one component which is Job Permutation (JP). We use array of integer, the length equals to L,
and each integer value equals the index of array of job correspondent [30]. From chromosome values we deduct three components the first is operations for each job in order to verify operation sequence constraint, the second is machine assignment and processing time for each operation as third. We start with first one, in fig.3 we can see operation[1]=1 because chromosome[1]=1 is the first instance of job 1 and operation[2]=1 because chromosome[2]=2 is the first instance of job 2 then operation[3]=1 because chromosome[3]=2 is the second instance of job 1 and so on. Now we have job and operation, we can choose randomly from table 1 one machine to complete the machine vector and time processing.

According to table 1 we can take these two examples:

**Chromosome 1 =**
```
[1 2 1 2 1]
```

**Operation sequence**
```
1 1 2 2 3
```

**Machine assignment randomly**
```
M1 M3 M4 M2 M4
```

**Processing time**
```
5 5 1 2 3
```

**Schedule 1 =**
```
O11 O21 O12 O22 O13
```

**Chromosome 2 =**
```
[1 2 1 1 2]
```

**Operation sequence**
```
1 1 2 1 2
```

**Machine assignment randomly**
```
M2 M3 M1 M3 M4
```

**Processing time**
```
2 5 8 4 1
```

**Schedule 2 =**
```
O11 O21 O12 O13 O23
```

**Figure 3. Two different permutations for chromosome with input data**

<table>
<thead>
<tr>
<th>M4</th>
<th>O22</th>
<th>O12</th>
<th>O13</th>
</tr>
</thead>
<tbody>
<tr>
<td>M3</td>
<td>O21</td>
<td></td>
<td></td>
</tr>
<tr>
<td>M2</td>
<td>O11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>M1</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| 1 2 3 4 5 6 7 8 9 |

**Figure 4. Gantt char for schedule 1**

These components must have the same length. Now we have all we need to calculate makespan. In this case we use some intermediate data structures such as processing which is matrix of \( n_j \) rows and no columns, and \( STime \) \( ETime \) presenting start and end time for each operation and finally \( EMachine \), end time for machines to calculate makespan for Schedule \( 1(O_{21}, O_{22}, O_{11}, O_{23}, O_{12}) \) we need the following matrix

\[
\text{Processing} = \begin{bmatrix}
    01 & 02 & 03 \\
    01 & 02 & 03 \\
\end{bmatrix}
\]

\[
\text{STime} = \begin{bmatrix}
    1 & 5 & 2 \\
    0 & 5 & 9 \\
\end{bmatrix}
\]

\[
\text{ETime} = \begin{bmatrix}
    01 & 02 & 03 \\
    01 & 02 & 03 \\
\end{bmatrix}
\]

\[
\text{EMachine} = \begin{bmatrix}
    M1 & M2 & M3 & M4 \\
    5 & 7 & 5 & 9 \\
\end{bmatrix}
\]

It is time to apply the genetic algorithm with selection, mutation and crossover operators on chromosome in our case is permutation of jobs.

We can not apply genetic operators on operations sequences and machine because we lose constraints and we will be obliged to add another algorithm to verify them.

The procedures of job permutation encoding chromosome and decoding used in this paper are described in Figure 5 and Figure 6

**Figure 5. Procedure of Job Permutation Encoding**

**Procedure:** Job Permutation Encoding

**Input:** Total number of state (machine) \( m \), total number of jobs \( n_j \), total number of operation for each job \( n_{o_j} \), processing time matrix \( Time \).

**Output:** Digit permutation chromosome \( CHROM; assign machine, processing time \).

**Step 1:** Generate randomly a vector \( CHROM \) of sequencing operations.

**Step 2:** Set index = 1.

**Step 3:** Choose randomly from \( Time \) matrix a machine such two operations cannot be processed simultaneously on the same machine.

Set machine(1) Set processing(1) Permutation; assignment machine and processing time for each operation is built and read from left to right. Setl = 1 + 1.

**Step 4:** repeat step 3 until \( l = K \)

Output \( CHROM, machine, processing \).

End

**Figure 6. Procedure of Job Permutation Decoding**

**Procedure:** Job Permutation Decoding

**Input:** Total number of state (machine) \( m \), total number of jobs \( n_j \), total number of operation for each job \( n_{o_j} \), digit permutation \( CHROM; assign machine, Machine, Time \).

**Output:** Best schedule, Makespan

**Step 1:** Deduct from \( CHROM \) operation’s order

**Step 2:** Choose machine for each operation

**Step 3:** Calculate Makespan

Best schedule, Makespan

End
4.3 Genetic operators

Selection operator

Choosing individuals for reproduction is the task of selection [28][30]. The chosen selection approach is adopted. Detailed steps are in figure 7.

**Procedure: Wheel Selection Job**

| Input: | Total number of state (machine) \( m \), total number of jobs \( n_j \), total number of operation for each job \( n_oj \), total number of individuals in one population \( n_p \), Makespan for each individual \( Mspan(n_p) \), Population |
| Output: | Selected population New population |

**Step 1:** Sort \( Mspan \); set index \( l = 1 \).

**Step 2:** \( \text{Prob}(l) = \frac{Mspan(l)}{\text{sum}(Mspan)} \)

**Step 3:** Repeat step 2 until \( l = n_p \)

**Step 4:** \( \text{Prob}_p = \text{cumsun}(\text{Prob}) \)

**Step 5:** Select best individuals

\( \text{Perc} = n_p \times 30\% \)

Keep 50% of best individuals

Set index \( k = 1 \), repeat

If \( \text{rdn} < \text{Prob}_p(k) \)

\( \text{Newpopulation}(l) = \text{Population}(k) \)

Break

**Endif**

**Until** \( k = \text{n}p \)

\( l = l + 1 \)

Until \( l = \text{Perc} \)

**Output:** New population With 30% of best individuals.

**End**

**Figure 7. Procedure of Wheel Selection Job**

Crossover operator

The goal of the crossover is to obtain better chromosomes to improve the result by exchanging information contained in the current good ones. In our work we carried out two kinds of crossover operator for the chromosomes.

In accordance with the adopted representation two crossover operators are used in this work [30]. Uniform crossover and a Precedence preserving order based crossover (POX). Uniform Crossover Operation is described in Figure 8.

**Procedure: Uniform Crossover Job**

| Input: | Total number of individuals in one population \( n_p \), Makespan for each individual \( Mspan(n_p) \), Newpopulation |
| Output: | Newpopulation |

**Step 1:** Choose randomly 2 individuals parent1; parent2

**Step 2:** Generate randomly a binary vector \( \text{unif}(n_p) \)

**Step 3:**

**Step 4:**

**Offset1(l) = parent1(l) Offset2(l) = parent1(l)**

**Endif**

**Until** \( k = \text{n}p \)

\( l = l + 1 \)

Until \( l = \text{Perc} \)

**Output:** New population

**End**

**Figure 8. Procedure of Uniform Crossover Job**

Mutation operator

Mutation introduces some extra variability into the population to enhance the diversity of population. Usually, mutation is applied with small probability. Large probability may destroy the good chromosome. In our research work we carried out one kind of mutation operator which is the mutation by values for the chromosome \( PJ \) (values mutation Job is described in Figure 9).

**Procedure: Values Mutation Job**

| Input: | Total number of individuals in one population \( n_p \), Makespan for each individual \( Mspan(n_p) \), Newpopulation |
| Output: | Newpopulation |

**Step 1:** Set index \( l = 1 \)

**Step 2:** Choose randomly an individual from Newpopulation

**Step 3:** Generate randomly 2 values in an individual

Permute these values if precedence order

**Step 4:** Repeat step 1 until 50% of population

**Output:** Newpopulation

**End**

**Figure 9. procedure of values mutation job**

5. Experimental results

To obtain meaningful results, we ran our algorithm five times on the same instance. The parameters used in the eGA were chosen experimentally to obtain a satisfactory solution within an acceptable time span. The effective GA is tested on Brandimarte’s data set (BR data). The data set consists of ten problems with number of jobs ranging from 10 to 20, number of machines ranging from 4 to 15 and number of operations for each job ranging from 5 to 15.


The proposed eGA algorithm for FSJ problem was coded in Matlab and run on 2.3 GHz PC with 4GO memory with the following parameters: popsize = 100, \( P_g = 0.8, P_m = 0.05 \), selection perc = 30%.

Table 2 summarizes the experimental results. It lists problem names, problem dimension (number of jobs \( \times \) number of machines), the best known solution (\( C_m \)), the solution obtained by our algorithm (eGA) and the solution obtained by each of the other algorithms. \( C_m \) denotes makespan time. *indicates the best known solution to date and denotes makespan time. ** indicate the best known solution best to date The computational results show that
the effective genetic algorithm is speed so far of searching optimal solution. Among the ten test problems, Mk01 could gain better solution in all the approaches. Mk03 and Mk08 could get the optimal solution in the first generation by using NGA. Mk02, Mk04, Mk05, Mk09 (Four problems) could gain the same good results as in M&G’s approach. One problem Mk06 could gain the same results as GENACE and two problems (Mk07, Mk10 problems) could gain the same results as Chen et al.

In figure 10-11 we draw the decrease of Makespan and Gantt chart for the Mk01 test problem with 10 jobs and 6 machines from BR’s data.

**Figure 10. Decreasing of the Makespan (MK01)**

**Figure 11. The Gantt chart of MK01**

5. Conclusion and future work

Recently, mono-objective flexible job-shop scheduling problem has attracted many researchers’ attention. Whereas this problem is well known as NP-hard. The complexity of this problem leads to the appearance of many heuristic approaches, and the research is mainly concentrated on evolutionary algorithms.

In this research, a meta-heuristic algorithm based on GA is developed for solving flexible job shop scheduling problems to minimize makespan. In our algorithm, we proposed a new chromosome representation scheme is applied, and efficient crossover and mutation operators are proposed to adapt. The numerical experiments indicate the effectiveness of the proposed approach. However, there are still a number of further works that need to be considered in the future. Can be considered for FJSP but not limited with this objectives. There may be some other side constraints and it can also be multi-objective optimization problem. Objective function can be to minimize the makespan, the tardiness, maximum lateness.

**Acknowledgment**

We would like to thank Pr. A. Driss for checking our English phrasing, Pr. Samai Aitouch for helping us with the organization and the methodology of this paper and finally my dear husband for his great support.

**References**


