WC-BEM 2012

A guide for genetic algorithm based on parallel machine scheduling and flexible job-shop scheduling

Bilgesu Ak a *, Erdem Koc b

a Bilgesu AK, Ondokuz Mayis University Industrial Engineering Department, Samsun-55139, Turkey
b Erdem KOÇ, Ondokuz Mayis University Mechanical Engineering Department, Samsun-55139, Turkey

Abstract

Parallel Machine Scheduling (PMS) and Flexible Job-shop Scheduling (FJS) are the hardest combinatorial optimization problems, they require very large scale search space. Solving this kind of combinatorial optimization problems with classical methods are almost impossible or takes considerable long time. Genetic Algorithms (GAs) have shown great advantages in solving combinatorial problems. GAs have the flexibility of set up different chromosome structures in case of distinctive scheduling problems. This paper presents a PMS and FJS chromosome structure, crossover and mutation operator from literature in order to guide for new researchers about scheduling with GAs.

Keywords: Parallel machine scheduling, flexible job shop problem, genetic algorithm, chromosome representation, crossover and mutation operators;

1. Introduction

In 1985, first Davis applied Genetic Algorithms (GAs) to scheduling problems. While applying GAs to scheduling problems the main problem is to find a suitable chromosome representation and genetic operators in order to create feasible schedules. Scheduling problems are in NP-hard class, so deterministic methods can not ensure short computation times or optimum solutions. This paper presents parallel machine scheduling (PMS), problem formulation, corresponding chromosome representation and genetic operators in second section. In third section flexible job-shop scheduling (FJS), problem formulation, corresponding chromosome representation and genetic operators are presented. In last section results are presented.
2. Parallel Machine Scheduling

2.1. Problem Formulation of Parallel Machine Scheduling

In parallel machine scheduling there are $m$ machines that can process all jobs in different or same speeds. Scheduling in parallel machines can be considered as a two step process. First, which jobs have to be allocated to which machines; second, the sequence of the jobs allocated to each machine.

2.2. Chromosome Representation

Chromosome representation form has vital effect on the GAs performance. Depending on the chromosomal representation and its related operators, generating feasible solutions and avoiding the use of repair mechanism can be provided. Kocamaz et al. suggested an approach to parallel machine scheduling with genetic algorithms developed from Cheng and Gen chromosome encoding method. An example for 3 machines and 5 jobs is given in Table 1. Every machine has different completion time for each job. For 3 machine–5 job scheduling, $3 \times 5 = 15$ bits exists in a chromosome representation in Fig. 1. For the first machine, there are 4 null values, because there are 5 possible null values (maximum number of jobs) and one of them is used by job 5.

<table>
<thead>
<tr>
<th>Machines</th>
<th>Jobs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Machine 1</td>
<td>Job 5</td>
</tr>
<tr>
<td>Machine 2</td>
<td>Job 4</td>
</tr>
<tr>
<td>Machine 3</td>
<td>Job 2</td>
</tr>
</tbody>
</table>

2.3. Crossover

Among the crossover types for permutation encoding, position based crossover (PBX) operator is used by Kocamaz et al. PBX needs a random selected parent string for deciding which genes must select for offspring. A set of first offspring’s positions from first parent is selected when parent string value is “1”. Each position of gene is independently marked with probability of 0.5. After that for filling empty genes, second parent’s genes were taken left from right in order without any repetition. The same procedure is applied for the second offspring, but procedure begins with parent two while dominant parent string value is “0”. In Figure 2, there is an example of PBX.

2.4. Mutation

For mutation operator, swapping method is used. In this method, two random genes are selected and swaps their positions. If two null values selected for swapping another selection must be done until a job is found. Offspring 1 and offspring 2 schedules are given in Table 2.

| J 5 | null | null | null | J 4 | J 1 | null | null | J 2 | J 3 | null | null | null |

Figure 1. Chromosome representation
3. Flexible Job-shop Scheduling

3.1. Problem Formulation of Flexible Job-shop Machine Scheduling

FJSP’s are computationally complex problems. Because FJSP’s are NP-hard –i.e., they can’t be solved within polynomial time or undirected search methods are not typically feasible for large scale problems. Among various search methods used for FJSPs the GA has been recognized as a general search strategy and optimization method. The important thing is how to create feasible schedules when applying FJSP to GA. Searchers developed various methodologies to create feasible schedules. Some of them set penalties on infeasible solutions, modify the breeding operators. A classic job shop scheduling problem has a set of \( n \) jobs processed by a set of \( m \) machines with the objective to minimize or maximize a criterion. Each job is processed on machines in a given order with a given processing time, and each machine can process only one type of operation. In FJSP, there are sets of machines with a number of identical machines in parallel. Each job has its own route as in JSP but job \( j \) is processed on any machine that is available among the set of that machine.

FJSP can be formulated generally as follows:
- \( J = \{ J_i \}, 1 \leq i \leq n \), the set of \( n \) jobs to be scheduled.
- \( O_{ij} \) is the operation \( j \) of \( J_i \).
- \( M = \{ M_k \}, 1 \leq k \leq m \), the set of \( m \) machines.
- \( p_{ij,k} \), the processing time of \( O_{ij} \), on machine \( M_k \).

Assumptions made in FJSP as follows:

Table 2. Production schedules for generated offspring 1 and 2 from parent 1 and 2

<table>
<thead>
<tr>
<th>Offspring</th>
<th>Machines-Job Sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Machines</td>
</tr>
<tr>
<td>Offspring 1</td>
<td>Machine 1</td>
</tr>
<tr>
<td></td>
<td>Machine 2</td>
</tr>
<tr>
<td></td>
<td>Machine 3</td>
</tr>
<tr>
<td>Offspring 2</td>
<td>Machine 1</td>
</tr>
<tr>
<td></td>
<td>Machine 2</td>
</tr>
<tr>
<td></td>
<td>Machine 3</td>
</tr>
</tbody>
</table>
• All machines are available at time 0;
• All job are ready to be processed at time 0;
• Each machine can process only one operation at a time;
• Each operation can be processed without interruption on one set of available machines;
• Recirculation occurs when a job visits a machine more than once;
• The sequence of operations for each job is predetermined and can not be changed.

A basic representation of flexible job shop scheduling is as follows in Table 3.

<table>
<thead>
<tr>
<th>Operations-Machines</th>
<th>Operations</th>
<th>M₁</th>
<th>M₂</th>
<th>M₃</th>
</tr>
</thead>
<tbody>
<tr>
<td>J₁</td>
<td>O₁,₁</td>
<td>3</td>
<td>3</td>
<td>xxx</td>
</tr>
<tr>
<td></td>
<td>O₁,₂</td>
<td>9</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>O₁,₃</td>
<td>xxx</td>
<td>7</td>
<td>5</td>
</tr>
<tr>
<td>J₂</td>
<td>O₂,₁</td>
<td>3</td>
<td>xxx</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>O₂,₂</td>
<td>4</td>
<td>8</td>
<td>xxx</td>
</tr>
</tbody>
</table>

Table 3 says that first operation \((O₁,₁)\) of first job \((J₁)\) can be processed on both \(M₁\) or \(M₂\), \(xxx\) denotes that \(O₁,₁\) can not be processed on \(M₃\). Similarly, second operation \((O₁,₂)\) of first job \((J₁)\) can be processed on \(M₁\), \(M₂\) or \(M₃\) and so on. The values in cells denotes the processing times. In FJSP, two decisions have to be considered. First is assignment of an operation to an approriate machine, second is sequencing the operations on each machine. In addition, for complex manufacturing systems, a job can visit a machine more than once called recirculation. These features of the FJSP significantly increase the complexity of finding even approximately optimal solutions.

3.2. Chromosome Representation

Ho et al. design a chromosomal representation generates feasible solutions that remains feasible under crossover and mutation. Their chromosomal representation (called OOMS) has two components: operation order and machine selection as seen in Figure 3.

Operation order component. They adopt the operation order representation from Cheng et al., Bierwirth and Varela et al.’s study. Consider the problem in Table 3. \(J₁\) has three operations and \(J₂\) has two operations. One possible schedule could be \((O₂,₁, O₁,₁, O₂,₂, O₁,₂, O₁,₃)\). However, when GA operators are applied to the schedule, infeasible schedules could exist. For instance, \((O₂,₂, O₁,₁, O₂,₁, O₁,₂, O₁,₃)\) is an infeasible schedule because \(O₂,₂\) can not be procesed before \(O₂,₁\). To avoid creating an infeasible solutions, an individual is obtained from this schedule by replacing each operation by the corresponding job index as seen in Fig 4.

| chromosome= | operation order | machine selection |

Figure 3. Structure proposed OOMS chromosome
Machine selection component. An array of binary values is used to present machine selection. For the problem in Table 3, one possible encoding of the machine selection part is shown in Fig 5.

From Table 3, it is understood that $O_{1,1}$ can be processed on $M_1$ or $M_2$, similarly $O_{1,2}$ can be processed on $M_1$, $M_2$ or $M_3$. Only one machine can be selected per operation. Practical results of this representation can be found in Ho and Tay’s study. Its empirical performance in comparison to other representations has also been shown to be very good.

3.3. Crossover

Because the OOMS chromosomal representation has two parts, crossover is applied on each part of the chromosome.

Operation order part: two-point crossover is applied. Consider two parents: (2 1 2 1 1) and (1 1 1 2 2). A substring is randomly selected from parent 1: (2 1 2 1 1). Operations in the selected string are $O_{1,1}$, $O_{2,2}$ and $O_{1,2}$ respectively. The corresponding positions of the characters in this string are then found and deleted in the second parent: (1 1 1 2 2). The substring is inserted to the second parent at the same position in the first parent to create a new child: (1 1 2 1 2).

Machine Selection part: two random numbers (for the two loci) are selected: $2 \leq r_1 \leq r_2 \leq (n-1)$, ($n$ is the length of the machine selection part). Two partial parts of the parents between the two loci are exchanged. Consider two machine selection parts of two parents in Figure 6. Let two random numbers are $r_1=2$ and $r_2=3$, than these two parts of two parents between position 2 and position 3 are exchanged as seen in Figure 7.

---

**Figure 4. OOMS: operation order component**

**Figure 5. OOMS: machine selection component**

**Figure 6. An example of two machine selection parts of two parents**
3.4. Mutation

Mutation is also applied on two parts of the chromosome.

Operation order: two random numbers \( r_1 \) and \( r_2 \) are generated such that \( 2 \leq r_1 \leq r_2 \leq (m-1) \), \( m \) is the length of the operation order part. The values in substring between these random \( r_1 \) and \( r_2 \) are inverted. Consider the chromosome \((21211)\), \( r_1=2 \) and \( r_2=3 \). The substring between position 2 and 3, \((21211)\), are inverted. After mutation, operation order part is \((22111)\).

Machine selection: First, operational memory is defined. Shortly, operational memory chooses the machine with the shortest processing time. The other suitable machines are updated after \( q \)th step by setting its suitability value to 1. For example, consider the operational memory for Table 3. \( O_{1,1} \) can be processed on \( M_1 \) or \( M_2 \). \( O_{1,2} \) can be processed on \( M_2 \) or \( M_3 \). \( O_{1,3} \) can be processed on \( M_3 \) and so on. If there are more than one machine suitable to process operation \( O_{i,j} \), the mutation operator would select a machine that is different from the current machine as given in Figure 8.

4. Results

Results gained by Kocamaz et al. and Ho et al. show that developed algorithms are more efficient than deterministic methods. Kocamaz et al. showed that the algorithm and the encoding method both perform successfully for achieving solution. The proposed developed encoding method can be easily applied to other parallel machine scheduling problems. Ho et al. showed that their algorithm is more efficient than the other approaches for solving the FJS, compared with GA approaches by Kacem et al., Mesghouni et al. and Tabu Search approach by Brandimarte. This paper is arranged in order to show the representation methods of GAs based on PMS and FJS which are very complex to schedule and compute with deterministic methods. The presented algorithms is aimed to guide for students and new researchers who are interested in GAs and scheduling.

References


