Flow Shop Scheduling Problem with Missing Operations: Genetic Algorithm and Tabu Search

R. Ramezanian*, M. Saidi-Mehrabad, D. Rahmani

Received: January 4, 2011 ; Accepted: August 25, 2011

Abstract Flow shop scheduling problem with missing operations is studied in this paper. Missing operations assumption refers to the fact that at least one job does not visit one machine in the production process. A mixed-binary integer programming model has been presented for this problem to minimize the makespan. The genetic algorithm (GA) and tabu search (TS) are used to deal with the optimization problem. According to computational experiments on data sets, it is suggested that GA is a more appropriate method to solve this problem. GA can reach good-quality solutions in short computational time, and can be used to solve large scale problems effectively.

Keywords Flow Shop Scheduling, Missing Operation, Mixed-Binary Integer Programming, Genetic Algorithm, Tabu Search.

1 Introduction

The flow shop scheduling problem (FSSP) is a well-known and complex combinatorial optimization problem which exists naturally in many real-life situations due to many practical as well as important applications for a job to be processed in series with more than one-stage in industry.

In the FSSP, we are given a set of jobs that have to be processed on a number of sequential machines. Each job has to be processed on all machines and the processing routes of all jobs are the same i.e., the operations of any job are processed in the same order. The job sequence of each machine has to be identified to minimize (or maximize) a specific performance measure (usually minimizing the makespan (M) or total flow time (TFT)). In the permutation FSSP, all jobs must process on machines in the same order, while in non-permutation FSSP, jobs can have different sequences on each machine.

Since Johnson (1954) [1] published his seminal paper on flow shop scheduling, it has remained a topic of interest for researchers and practitioners. There are several mathematical models for formulation of flow shop scheduling (for instance, see [2-8]). Stafford and Tseng [3-5,8] widely studied this context and several integer programming models for the flow shop scheduling problems. Their research focuses only on the regular permutation flow shop problem. Ziaee and Sadjadi [7] proposed some mathematical models for the general form of flow shop scheduling problem.
In the real world environment it is possible that a job is not processed on all machines (i.e. having some missing operations). Glass et al. [9] consider flow shop scheduling with missing operations (FSSPMO) for two machines. They [10] also investigate the no-wait scheduling of n jobs in a two-machine flow shop, where some jobs require processing on the first machine only. The objective is to minimize the maximum completion time, or the makespan. Leisten and Kolbe [11] consider scheduling jobs with a missing operation in permutation flow shops. Sadjadi et al. [12] present some models for general flow shop scheduling with a missing operation assumption. However, they just use an optimization solver to deal with some small size problems. We note that this method (using a solver) is not applicable for medium and large size problems due to the fact that FSSP is NP-hard [13]. The computational complexity of a more general problem (admitting missing operations) may be much harder than that of the corresponding problem without missing operations.

Exact algorithms for the FSSP failed to achieve high-quality solutions for problems of large size in reasonable computational time and, thus, some researchers focused on heuristic methods (for instance, see [14-16]). Recently, because of high efficiency of meta-heuristic algorithms (e.g., simulated annealing, tabu search, genetic algorithms), there are a lot of interests in using them (for instance, see [17-19]).

In this paper, we present a mixed-binary integer programming model flow shop scheduling problem with missing operations for the objective of minimizing the makespan. Due to the complexity of this problem, the genetic algorithm (GA) and tabu search (TS) are used to deal with the optimization.

The remainder of the paper is organized as follows: Section II defines the concept of "missing operations". In III we present our mathematical model for FSSP with missing operations. Section IV and V describe the genetic algorithm and tabu search for solving our proposed model. Section VI presents the computational results acquired and, finally, the final section provides conclusions and suggestions for future research.

2 Missing operations

The flow shop scheduling problem with missing operations can be described as follows: Each of n jobs from set $J=\{1,2,...,n\}$ will be sequenced through m machines ($i=1,2,...,m$). Job $j \in J$ has a sequence of $l_j$ operations through a subset of $m$ machines (jobs may have zero processing time on some machines). Operation $O_{ij}$ corresponds to the processing of job $j$ on machine $i$ during an uninterrupted processing time $t_{ij}$. At any time, each machine can process at most one job and each job can be processed on at most one machine.

3 Model formulation

In this section, a mathematical model for FSSP with missing operations similar to [7,12] is presented.

A. Assumptions

- A job has some operations that each of them is to be performed on a specified machine. Some jobs may not process on some machines so the processing time of them on that machine is zero (missing operations).
• Machines cannot process two jobs at the same time.
• Each job is processed on at most one machine at a time.
• Setup times for the operations are sequence-independent and are included in processing times.
• Jobs are allowed to wait between two stages, and the storage is unlimited.
• There is only one of each type of machine.
• No more than one operation of the same job can be executed at a time.
• All programming parameters are deterministic and there is no randomness.

B. Parameters

\( n \): Number of jobs
\( m \): Number of machines
\( i \): Machine index, \((i=1,\ldots,m)\)
\( j \): Job index, \((j=1,\ldots,n)\)
\( k \): Order index, \((k=1,\ldots,n)\)
\( t_{ij} \): Processing time of job \( j \) on machine \( i \)
\( \delta_{ij} \): Binary parameter taking value 1 if the job \( j \) is not processed on machine \( i \) and 0 otherwise.

C. Decision variables

\( Z_{ij} \): Completion time of job \( j \) on machine \( i \) if this job does not need \( i \)th machine. Otherwise, the earliest possible time for starting job \( j \) on machine \((i+1)\).
\( q_{ijk} \): Completion time of job \( j \) on machine \( i \) in \( k \)th order if this job is processed on \( i \)th machine, otherwise it is meaningless.
\( x_{ijk} \): Binary variable taking value 1 if the job \( j \) on machine \( i \) is processed in \( k \)th order and 0 otherwise.

The mathematical model to minimize the maximal completion time of all jobs (makespan) is as follows:

\[
\text{Min } y
\]

s.t.

\[
y \geq Z_{mj}, \quad \forall j
\]

\[
\sum_{k=1}^{n} q_{ijk} x_{ijk}, (Z_{ij} = 0) \quad \forall i, j
\]

\[
Z_{ij} = \delta_{ij} Z_{(i-1)j} + (1-\delta_{ij}) \sum_{k=1}^{n} q_{ijk} x_{ijk}, (Z_{ij} = 0), \quad \forall i, j
\]

\[
\sum_{ij} (q_{ijk} - t_{ij}) x_{ijk} \geq Z_{(i-1)j}, (Z_{ij} = 0) \quad \forall i, j
\]

\[
\sum_{j=1}^{n} (q_{ijk} - t_{ij}) x_{ijk} \geq \sum_{j=1}^{n} q_{ijk} x_{ijk}, (Z_{ij} = 0), \quad \forall i, j
\]

\[
\sum_{j=1}^{n} x_{ijk} = 1, \quad \forall i, k
\]

\[
\sum_{k=1}^{n} x_{ijk} = 1, \quad \forall i, j
\]

\[
q_{ijk} \leq M x_{ijk}, \quad \forall i, j, k
\]

\[
q_{ijk} \geq 0, \quad \forall i, j, k
\]

\[
x_{ijk} \in \{0, 1\}, \quad \forall i, j, k
\]

The objective function (1) considers the minimization of the makespan. The constraint set (2) ensures that the makespan is equal to the maximum completion time of any jobs. If the job \( j \)
is processed on the last machine, $Z_{mj}$ determines completion time of job $j$; Otherwise, $Z_{mj}$ refers to the completion time of last operation of this job on one of the previous machines. The constraint set (3) determines the completion time of job $j$ on machine $i$. If the job $j$ is not processed on machine $i$ ($t_{ij}=0$), $Z_{ij}$ equals to the completion time of job $j$ on the previous machine. The constraint sets (4) and (5) insure that a job does not start on a machine until it finishes processing on the previous machine and its predecessor has completed processing on that machine. The constraint set (6) insures that in each machine, each sequence position is filled with only one job and the completion time variables. Based on this constraint set, if each binary variable is equal to zero then its completion time variable will be equal to zero. (9) and (10) are logical constraints.

We should remind that in the case of having missing operations, the makespan is not necessarily determined by the last machine. However, in the proposed model completion time of last operation of each job that may be on any machine is transferred to the last machine by using constraint (3). Thus, we can examine just the last machine for the makespan.

4 The genetic algorithm

The GA was proposed by Holland [20] to encode the factors of a problem by chromosomes, where each gene represents a feature of the problem. The combinations of genes are evolved through the genetic operators so that the chromosomes would approach the optimal solution generation by generation. Our implementation of genetic algorithm is presented as follows.

D. Design of Genes

In this paper, each gene is a job and the chromosome is a job sequence vector on machines. We first assume that all jobs have an order on any machine which is selected randomly. Jobs that have zero processing time on a machine are also assigned an order which is not a real order and does not have precedence constraint (starting time of this order is not restricted by finishing time of previous order). Consider a flow shop scheduling problem with missing operation with 5 jobs and 3 machines (see Table 1).

<table>
<thead>
<tr>
<th>Job</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>

The job sequence for this example represented in Figure 1 can be translated into a list of ordered jobs below:
Flow Shop Scheduling Problem with …

Machine I: \( j_2 > j_1 > j_3 > j_5 > j_4 \)

Machine II: \( j_4 > j_1 > j_2 > j_3 > j_5 \)

Machine III: \( j_5 > j_3 > j_4 > j_2 > j_1 \)

<table>
<thead>
<tr>
<th>Priority (k)</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Job sequence on machine 1: ( V_1(k) )</td>
<td>2</td>
<td>1</td>
<td>5</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>Job sequence on machine 2: ( V_2(k) )</td>
<td>4</td>
<td>1</td>
<td>2</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>Job sequence on machine 3: ( V_3(k) )</td>
<td>3</td>
<td>2</td>
<td>4</td>
<td>5</td>
<td>1</td>
</tr>
</tbody>
</table>

*Highlight jobs on a machine have zero processing time

Fig. 1 Illustration of the job sequence vector on machines

E. The genetic operators

1) Crossover

Crossover operator recombines two chromosomes to generate a number of children. Offspring of crossover should represent solutions that combine substructures of their parental solutions. In this paper, the enhanced order crossover expanded from the classical order crossover has been applied and works as follows:

**Step 1.** Randomly choose two chromosomes, named parent 1 and parent 2:

**Step 1.1.** Two chromosomes are selected randomly, then the chromosome with lower fitness function is chosen and named parent 1.

**Step 1.2.** Repeat Step 1.1, then name the chromosome with lower fitness function parent 2.

**Step 2.** Do the following steps for the same machine in selected chromosomes (parent 1 and parent 2):

**Step 2.1.** Randomly select a subsection of job sequence for \( i \)th machine from parent 1.

**Step 2.2.** Produce a proto-child by copying the substring of job sequence into the corresponding positions.

**Step 2.3.** Starting with the first position from \( i \)th machine of parent 2, delete the jobs which are in the substring from \( i \)th machine of the second parent. The resulted sequence of jobs contains the jobs that the proto-child needs.

**Step 2.4.** Place the remaining jobs into the empty positions of the proto-child from left to right according to the order of the sequence in the \( i \)th machine of second parent.

The procedure is illustrated in Fig 2.

Fig. 2 Illustration of the enhanced order crossover

2) Mutation

Our mutation mechanism (Swap operator) works as follows:

**Step 1.** Randomly choose one chromosome.

**Step 2.** Do the Following steps for the same machine in the selected chromosome:

**Step 2.1.** Randomly choose two priorities from \( i \)th machine of selected chromosome in step 1.
Step 2.2. Replace the selected jobs with each other.
The procedure is illustrated in Figure 3. The mutation rate is considered 0.05.

![Fig. 3 Illustration of the mutation](image)

F. Fitness function
The fitness function is the same as the objective function which is defined in section 3, namely the makespan. In the proposed genetic algorithm the lower fitness function is desired.

\[ C_j: \text{Completion time of job } j \]

\[ \text{Fitness Function: } C_{\text{max}} = \max_{j} C_j \]

The genetic algorithm used in this paper is illustrated in Figure 4.

![Fig. 4 Flowchart of the proposed GA](image)

5 Tabu search (TS)
The TS method is iterative, a neighborhood based search method. In this technique, at each iteration, a move is performed to the best solution in the neighborhood of actual one. To avoid cycling and to escape from local optimum, the memory of visited solutions is introduced. The most frequently used type of memory is a short-term memory called the tabu list. In the tabu list some number of recently visited solutions, their attributes, or moves leading to them are
stored. The move is tabu (omitted in the search process) if it leads to the solution already visited [21,22]. In the following we present our implementation of TS.

G. Initial solution
The best solution among solutions generated randomly is selected as an initial solution.

H. Tabu list
In the tabu list, criterion values of maximum number of recently visited solutions are stored. The list is initialized with empty elements. A newly added element replaces the oldest one. Job sequence and Makespan of each solution is stored in the tabu list. The move is tabu if it leads to the solution with criterion value equal to the one of the stored in the tabu list. For our experiments we accepted the tabu list with the fixed length. $MaxL = \max\{n, m\}$.

I. Neighborhood search scheme
Since the time required to evaluate the entire neighborhood increases very fast with the increase of the problem size, we examine only part of the neighborhood. In our algorithm we evaluate only $m \times n$ random neighbors. Swap operator is used for the neighborhood scheme.

J. Stopping Condition
The search process stops if the number of iterations is greater than maximum number of iterations, an priori fixed constant. In our experiments we accepted $Max$-Iteration$=1000$.

6 Computational results
The mathematical model for FSSP with missing operations is solved by the genetic algorithm and tabu search as well as LINGO8. The genetic algorithm and tabu search are coded in MATLAB R2007(b) and all tests are conducted on a Pentium_IV PC at 2.4 GHz with 1.0GB of RAM. We generate random problem instances for number of jobs and number of machines from $2 \times 2$ to $50 \times 50$. Job processing times on each machine are drawn from discrete uniform distribution in the interval $[0–5]$.

For small size problems, the genetic algorithm, tabu search and LINGO can solve them in a short time but as the size of problem increases the computation time of LINGO goes up exponentially. For example for a $5 \times 5$ problem, it takes 24 hours for LINGO. The comparison for small-sized problems between genetic algorithm, tabu search and LINGO is shown in Table 2.
Table 2 Comparison results of GA, TS and LINGO for small size problems

<table>
<thead>
<tr>
<th>Prob.</th>
<th>m</th>
<th>n</th>
<th>GA (Popsize=100, Iter=20) C_{max} Time (S)</th>
<th>TS (Max-Iter=100) C_{max} Time (S)</th>
<th>Lingo C_{max} Time (S)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>2</td>
<td>8</td>
<td>0.11</td>
<td>8</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>3</td>
<td>11</td>
<td>0.15</td>
<td>11</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>4</td>
<td>14</td>
<td>0.18</td>
<td>14</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>5</td>
<td>15</td>
<td>0.21</td>
<td>15</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>2</td>
<td>11</td>
<td>0.16</td>
<td>11</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>3</td>
<td>14</td>
<td>0.21</td>
<td>14</td>
</tr>
<tr>
<td>7</td>
<td>3</td>
<td>4</td>
<td>18</td>
<td>0.27</td>
<td>18</td>
</tr>
<tr>
<td>8</td>
<td>4</td>
<td>2</td>
<td>16</td>
<td>0.22</td>
<td>16</td>
</tr>
<tr>
<td>9</td>
<td>4</td>
<td>3</td>
<td>24</td>
<td>0.33</td>
<td>24</td>
</tr>
<tr>
<td>10</td>
<td>5</td>
<td>2</td>
<td>18</td>
<td>0.25</td>
<td>18</td>
</tr>
<tr>
<td>11</td>
<td>5</td>
<td>3</td>
<td>19</td>
<td>0.38</td>
<td>19</td>
</tr>
<tr>
<td>12</td>
<td>5</td>
<td>5</td>
<td>22</td>
<td>0.54</td>
<td>22</td>
</tr>
</tbody>
</table>

* This problem is interrupted

The genetic algorithm and tabu search can solve the problems in a short time efficiently. For large scale problems, the results of GA and TS are presented in Table 3. It is obvious that GA can produce better solutions than TS. Furthermore, the genetic algorithm has the ability to reach a stable solution which is depicted in figures 5 and 6.

Table 3 The results of GA and TS for large scale problems

<table>
<thead>
<tr>
<th>Prob.</th>
<th>m</th>
<th>n</th>
<th>GA C_{max} Time (s)</th>
<th>TS C_{max} Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10</td>
<td>10</td>
<td>102</td>
<td>9.84</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>15</td>
<td>156</td>
<td>13.82</td>
</tr>
<tr>
<td>3</td>
<td>10</td>
<td>20</td>
<td>221</td>
<td>18.56</td>
</tr>
<tr>
<td>4</td>
<td>15</td>
<td>10</td>
<td>127</td>
<td>12.87</td>
</tr>
<tr>
<td>5</td>
<td>15</td>
<td>15</td>
<td>226</td>
<td>20.11</td>
</tr>
<tr>
<td>6</td>
<td>15</td>
<td>20</td>
<td>344</td>
<td>27.98</td>
</tr>
<tr>
<td>7</td>
<td>20</td>
<td>10</td>
<td>222</td>
<td>20.17</td>
</tr>
<tr>
<td>8</td>
<td>20</td>
<td>15</td>
<td>293</td>
<td>27.09</td>
</tr>
<tr>
<td>9</td>
<td>20</td>
<td>20</td>
<td>420</td>
<td>37.78</td>
</tr>
<tr>
<td>10</td>
<td>20</td>
<td>30</td>
<td>786</td>
<td>59.65</td>
</tr>
<tr>
<td>11</td>
<td>30</td>
<td>30</td>
<td>1212</td>
<td>90.77</td>
</tr>
<tr>
<td>12</td>
<td>50</td>
<td>50</td>
<td>3908</td>
<td>639.42</td>
</tr>
</tbody>
</table>

* This problem is interrupted
In this paper, non-permutation flow shop scheduling problem with missing operations which is often occurring in shop of real world is considered. Regarding to the complexity of this problem, we implemented the genetic algorithm and tabu search for solving this problem. Computational experiments have been performed and demonstrated that GA is a more appropriate method to apply in this problem in comparison with TS and can reaches good-quality solutions in short computational time.

References