A parallel genetic algorithm for the job shop scheduling problem

Nguyen Huu Mui
Department of Information Technology
Hanoi University of Education
Hanoi, Vietnam
muinh@hnu.edu.vn

Vu Dinh Hoa
Department of Information Technology
Hanoi University of Education
Hanoi, Vietnam
hoavudinh@gmail.com

Luc Tri Tuyen
Institute of Information Technology
Vietnam Academy of Science and Technology
Hanoi, Vietnam
tuyenlt@ioit.ac.vn

Abstract—This paper presents a parallel genetic algorithm for the job shop scheduling problem (JSP). There are following innovations in this new algorithm: active schedules are created by the priority rules of Giffler and Thompson; the mutation uses neighborhood searching techniques; the crossover uses GT algorithm and is performed on 3 parents. The convergence of the algorithm is also shown in this paper.

We illustrate this new method on the parameters of Muth and Thompson’s benchmark problems. It can produce optimal solutions at a high percentage of accuracy. Our proposed method is preeminent in comparison with other methods on both the calculation time and the speed of finding optimal solutions.

Keywords—jobshop scheduling; parallel genetic algorithm.

I. INTRODUCTION

The most typical model of scheduling is the Job Shop Scheduling Problem (JSP). The JSP plays a very important role both in theory and practice. Apparently, the JSP was formulated and investigated for the first time by Akers & Friedman [1]. In fact, in Soviet literature, the JSP is usually called the Akers - Friedman problem or the AF problem for short.

A general JSP can be described as follow:
Given a set of \( n \) jobs \( \{J_i\}_{i=1}^{n} \) which is processed on a set of \( m \) machines \( \{M_j\}_{j=1}^{m} \), the problem can be characterized as below:
1. Each job must be processed on each machine following a given order called technological sequence of machines.
2. Each machine is only able to process only one job at a moment.
3. The operation of job \( J_i \) is processed on machine \( M_j \) is called to be the operation \( O_{ij} \).
4. The processing time of \( O_{ij} \) is denoted by \( p_{ij} \).
5. The starting time and complete time of an operation \( O_{ij} \) is denoted by \( s_{ij} \) and \( c_{ij} \) respectively.
6. The time needed to complete all the jobs is called to be the makespan, which is denoted by \( C_{max} \). By definition, \( C_{max} = \max \{ c_{ij} \}_{i=1}^{n} \}_{j=1}^{m} \).

The JSP problem is one of the most difficult combinatorial optimization problems up to now. In fact, we only have been known few of solvable polynomial cases of the JSP. All other cases of JSP are NP-hard, this is such the research result of Sotskov & Shaklevich [7]. These complex results are summarized in Table 2.

Table 1. JSP with 3 jobs and 3 machines

<table>
<thead>
<tr>
<th>Job</th>
<th>Machine (processing time)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1 (4) 2 (4) 3 (4)</td>
</tr>
<tr>
<td>2</td>
<td>1 (3) 3 (4) 2 (5)</td>
</tr>
<tr>
<td>3</td>
<td>2 (4) 1 (3) 3 (2)</td>
</tr>
</tbody>
</table>

An example of the JSP with 3 jobs and 3 machines is given in Table 1. The data includes the technological sequence of machines for each job together with the processing times in parentheses. The problem is equivalently represented by a job sequence matrix \( \{T_{ik}\} \) and a processing time matrix \( P=[p_{ik}] \) as follow:

\[
\{T_{ik}\} = \begin{pmatrix}
1 & 2 & 3 \\
1 & 3 & 2 \\
2 & 1 & 3
\end{pmatrix}, \quad \{p_{ik}\} = \begin{pmatrix}
4 & 4 & 4 \\
3 & 4 & 5 \\
4 & 3 & 2
\end{pmatrix}
\]

Table 2. The computational complexity of the JSP

<table>
<thead>
<tr>
<th>Number of jobs ( n )</th>
<th>2 ( P )</th>
<th>3 ( P )</th>
<th>constant ( P )</th>
<th>arbitrary ( P )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of machines ( m )</td>
<td></td>
<td>2</td>
<td>3</td>
<td>constant</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>P</td>
<td>NP-hard</td>
<td>NP-hard</td>
</tr>
<tr>
<td>constant</td>
<td></td>
<td>P</td>
<td>NP-hard</td>
<td>NP-hard</td>
</tr>
<tr>
<td>arbitrary</td>
<td></td>
<td>NP-hard</td>
<td>NP-hard</td>
<td>NP-hard</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Number of jobs ( n )</th>
<th>2 ( P )</th>
<th>3 ( P )</th>
<th>constant ( P )</th>
<th>arbitrary ( P )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of machines ( m )</td>
<td></td>
<td>2</td>
<td>3</td>
<td>constant</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>P</td>
<td>NP-hard</td>
<td>NP-hard</td>
</tr>
<tr>
<td>constant</td>
<td></td>
<td>P</td>
<td>NP-hard</td>
<td>NP-hard</td>
</tr>
<tr>
<td>arbitrary</td>
<td></td>
<td>NP-hard</td>
<td>NP-hard</td>
<td>NP-hard</td>
</tr>
</tbody>
</table>

Genetic Algorithms (GAs) model biological processes to optimize highly complex objective functions. They allow a population composed of many individuals to evolve under specified selection rules to a condition that maximizes the
“fitness”. This method was developed by John Holland [8] over the course of the 1960s and 1970s, and popularized by one of his students, David Goldberg who successfully applied a GA to the control of gas-pipeline transmission. Genetic Algorithms have been used successfully in various fields of computer science, including machine learning, control theory and combinatorial optimization. GAs can be uniquely characterized by their populations based on searching strategies and their operators: mutation, selection and crossover. Nakano and Yamada [9] were among the first researchers who applied a conventional GA using binary representation of solutions into the job shop scheduling problem. After that, they proposed a GA using the representation of solutions together with crossover and mutation, which was based on the GT algorithm.

Recently, many researchers have used GA to combine with other techniques to create hybrid methods to address the JSP such as Lee HuiPeng et al [10], F. Guerriero [11], Rui Zhang et al [12], Yamada [13] etc.

Although a number of works has been in progress in recent years, all proposed methods cannot address the JSP completely up to now.

This paper proposes a new genetic algorithm for the JSP. This new algorithm has following innovations:

1. The prior-rule proposed by Giffler and Thompson is used to create active schedules.
2. The mutation uses neighborhood searching technique.
3. The crossover uses GT algorithm [14] and is implemented on 3 parents.
4. Selection operation is composed of the following two steps:

   Step 1: Select an individual which has the best fitness in the population at generation t-th.
   Step 2: Select (n-1) individuals using the principle “Roulette Wheel”.

II. THE HYBRID GENETIC ALGORITHM FOR THE JSP

A. Solution coding

We suppose that there are n jobs can be processed on m machines. The number of operations of job $J_i$ is denoted by $\text{job}[i]$ ($\text{job}[i]$ ≤ m, for all i). The sum of operations to process all the jobs are $L= \sum_{i=1}^{n} \text{job}[i]$. We encode the operations of $J_i$ from 1 to $\text{job}[1]$, of $J_2$ from $\text{job}[1]+1$ to $\text{job}[1]+\text{job}[2]$,…, of $J_n$ from $\text{job}[1]+\text{job}[2]+ ... + \text{job}[n-1]+1$ to L. Thus, each solution is a permutation of L natural numbers $\{1, 2, 3, ..., L\}$ satisfying the constraints of the problem.

For example, the problem with 3 jobs and 3 machines is given in Table 1. Operations are encoded by natural numbers as in Table 3.

According to the matrix $\{T_{jk}\}$, operations 1, 4, 8 are processed on machine 1. Thus, the codes on $M_1$ are a permutation of the set $\{1, 4, 8\}$. Similarly, the codes on $M_2$ are a permutation of the set $\{2, 6, 7\}$, and on $M_3$ are a permutation of the set $\{3, 5, 9\}$. A valid solution for the problem may be as Figure 1.

![Figure 1. A valid solution for the 3 × 3 job shop problem](image)

This solution can be shown by a solution matrix $S_k$. Where, $S_{jk}$ = i, it means the $k$th operation on machine $M_i$ is job $J_i$.

$$
\{S_k\} = \begin{bmatrix}
1 & 2 & 3 \\
3 & 2 & 1 \\
2 & 1 & 3
\end{bmatrix}
$$

**B. Generate a set of initial solutions**

The JSP is represented by the technological sequence matrix $\{T_{jk}\}$, and the processing time matrix $\{p_{jk}\}$. A set of initial active schedules for the problem can be generated by using the GT algorithm proposed by Giffler & Thompson [14].

**C. Construct the objective function**

The objective function is represented as follow:

$$\text{Fitness} = M - C_{\text{max}}$$

where $C_{\text{max}}$ is makespan of the solution, $M$ is a given parameter to change the minimum problem to the maximum problem (because the genetic algorithm only applies immediately for the max problems).

**D. Mutation operator**

The mutation operator is composed of the steps as follow:

1. Select randomly one operation (ope1) of the parent. Identify the machine ($M_{\text{ope1}}$) which performs this operation and identify the position (pos1) of this operation.
2. Select randomly one operation (ope2) of the parent. Identify the machine ($M_{\text{ope2}}$) which performs this operation and identify the position (pos2) of this operation.
3. If $M_{\text{ope1}} = M_{\text{ope2}}$ then perform mutation (make a permutation between these two positions). The acquired result is a child chromosome. Otherwise, the parent chromosome is kept intact.

---

Table 3. The operations are encoded by natural numbers

<table>
<thead>
<tr>
<th>Jobs</th>
<th>Operation coding</th>
</tr>
</thead>
<tbody>
<tr>
<td>$J_1$</td>
<td>1 2 3</td>
</tr>
<tr>
<td>$J_2$</td>
<td>4 5 6</td>
</tr>
<tr>
<td>$J_3$</td>
<td>7 8 9</td>
</tr>
</tbody>
</table>

---
4. For all operations of the child chromosome, update its starting and complete time.

5. The child chromosome is only accepted if its fitness better than its parent’s fitness (mutation operator can be repeated a number of times). Otherwise, the parent chromosome is kept intact.

E. Crossover operator

The crossover operator combines the homogenous crossover with GT algorithm and it is performed on three parents called $p_1$, $p_2$, and $p_3$. These parents are shown by correlative solution matrices $S^1 = \{ S^1_{ik} \}$, $S^2 = \{ S^2_{ik} \}$ and $S^3 = \{ S^3_{ik} \}$. Genes of offspring $p = \{ S_p \}$ are the ones getting from its parents. The crossover operation is described below:

1. Initialize $G$ as a set of operations which will be the first operations in the technological sequence, i.e., $G = \{ O_{IT1}, O_{IT2}; \ldots, O_{ITn} \}$. For each operation $O \in G$, let $ES(O) := 0$ and $EC(O) := p(O)$.

2. Find the operation completed first $O_j \in G$. A subset of $G$ that consists of the operations processed on machine $M_j$ is denoted by $G_j$.

3. Calculate the competitive set $C[M_j, k] \subset G_j$, where $k - 1$ is the number of operations already scheduled on $M_j$.

4. Select one of the parents $\{ p_1, p_2, p_3 \}$ called $p$ corresponding to the value of matrix $H_{jk}$ such that $p := p_{H_{jk}}$ and $S^p := S^H$. For each $O_j \in C[M_j, k]$ with the job $i$, there exists an index $l$ such that $l_i = i$. Let $l_n$ be the smallest index among them, i.e., $l_n = \min \{ l \mid S_j = l \}$ and $O_j \in C[M_j, k]$ then let $r := S_{j_{l_n}}$. The operation $O_j \in C[M_j, k]$ then is selected to schedule for $p$ earliest among the other members of $C[M_j, k]$.

5. Schedule $O_j$ is as the $k^{th}$ operation on $M_j$, i.e. $S_{j_k} := r$, with its starting and completion times equal to $ES(O_j)$ and $EC(O_j)$ respectively: $s(O_j) = ES(O_j)$; $c(O_j) = EC(O_j)$.

6. For all $O_j \in G_j \setminus \{ O_j \}$, update:
   - $ES(O_j) := \max \{ ES(O_j), EC(O_j) \}$
   - $EC(O_j) := ES(O_j) + p(O_j)$

7. Remove $O_j$ from $G$ (and therefore from $G_j$) and add operation $O_n$ which is next to $O_j$ in the technological sequence to $G$ if such $O_n$ exists, i.e., if $j = T_d$ and $k < m$, then $s := T_{d+1}$ and $G := (G \setminus \{ O_j \}) \cup \{ O_n \}$.

8. Repeat from step 1 to step 7 until all operations are scheduled.

9. The output solution matrix $\{ S_p \}$ is an active schedule obtaining from the set of starting and completion times $\{ s(O_j) \}$ and $\{ c(O_j) \}$ respectively, where $i = S_{jk}$.

F. Selection operator

Selection operator is composed of the following steps:

Step 1: Select an individual which has the best fitness in the population at generation $t$-th.

Step 2: Build up the intermediate solution set $P(t)$ by:
   - perform the mutation operator for $P(t)$, we get $P_1(t)$
   - perform the crossover operator for $P(t)$, we get $P_2(t)$
   - $P(t) = P(t) \cup P_1(t) \cup P_2(t)$

Step 3: Select $(n - 1)$ individuals randomly based on the principle “Roulette Wheel”.

G. Evolutionary algorithm

The evolutionary algorithm for the job shop problem is described as follow:
Procedure GA_JSP
Begin
\[ t = 0 \]
Initialize \( P(t) \)
Evaluate \( P(t) \)
Calculate the evolution function \( \text{Eval}(P(t)) \)
While not meet the termination condition, do
Begin
\[ t = t + 1 \]
Select \( P(t) \) from \( P(t-1) \)
Evaluate \( P(t) \)
Calculate the evolution function \( \text{Eval}(P(t)) \)
If \( \text{Eval}(P(t-1)) \geq \text{Eval}(P(t)) \) then \( t = t - 1 \)
End
End.

III. THE PARALLEL GENETIC ALGORITHM FOR THE JOB SHOP SCHEDULING PROBLEM

A. Description of the algorithm

For the algorithm presented in section II, we see that there are a lot of tasks which are computed independently, especially in the mutation and crossover.

In the parallel environment, suppose that we have \( S \) individuals in the population, and \( N \) is the number of processors running in parallel. The model used in this method is the master-slave model, in which a processor is as the master processor while \( N-1 \) remaining processors are as the slaves. The parallel algorithm inherits the methods of coding, initialization and evolutionary operators in the sequential algorithm. The parallel algorithm has some differences as below:

- At each of the iterations, the tasks of the master and the slaves are listed in Table 4.

<table>
<thead>
<tr>
<th>Master</th>
<th>Slaves (1&lt;i&lt;N-1)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Initialize the initial population, perform the evolutionary operators and select the optimal result from ( S ) individuals.</td>
</tr>
<tr>
<td></td>
<td>Receive the optimal results from the Slaves, calculate the whole optimum.</td>
</tr>
<tr>
<td></td>
<td>Send the optimal result to the Master.</td>
</tr>
</tbody>
</table>

- All tasks of the sequential genetic algorithm are parallelized. They are performed on \( N-1 \) processors at the same time.

- Since the structure of the sequential algorithm is not broken, the quality of the solution after parallelizing is as good as the one of the sequential algorithm.

- The algorithm does not require complex topologies.

The functions and procedures in the program simulating the parallel genetic algorithm mostly inherit from the program which simulates the sequential genetic algorithm.

B. The parallel genetic procedure for the JSP

Input: The number of threads opened corresponding to the number of machines, the number of generations, number of individuals in the population, the probability of mutation, and the probability of crossover.

Output: An array to store the optimal results getting from the Slaves and the Master.

Procedure PGA_JSP
Begin
Open the bandwidth and initialize the threads;
Slaves: auto-initialize the initial population;
\[ t \leftarrow 0 \]
Initialise \( P(t) \)
Evaluate \( P(t) \)
While (not meet the termination condition) do
Begin
Build up the solution set \( P'(t) \) by applying the mutation and crossover operation:
\[ + \text{ Applying mutation for } P(t), \text{ obtain } P_1(t) \]
\[ + \text{ Applying crossover for } P(t), \text{ obtain } P_2(t) \]
\[ P'(t) = P(t) \cup P_1(t) \cup P_2(t) \]
Evaluate \( P'(t) \)
\[ t \leftarrow t + 1 \]
Perform selection for \( P(t) \) from \( P'(t-1) \)
End
The Slaves select the optimal results and send them to the Master;
The Master finds the whole optimal result;
End.

IV. THE EMPIRICAL RESULTS

Based on the proposed method in this article, we have set up a program using the parallel programming library called MPI with C++ source code and run on the CSS computer server system with 6 computers, in which, each computer has the pentium-4 CPU with 4GB free of ram. The input data we used for our program is the parameters borrowing from the benchmark problems, which proposed by Muth and G.L.Thompson [15]. These are the famous test problems difficult to solve but have known the optimal results.

The set of parameters and the empirical results for the program we proposed are shown in Table 5. The optimal result of mt06 problem was found rapidly only after several times of test. For the mt10 problem, the optimal result was found after every 20 times of run in average.
Table 5. The empirical results of our method

<table>
<thead>
<tr>
<th>Problem</th>
<th>Solution size</th>
<th>Number of generations</th>
<th>pc</th>
<th>pm</th>
<th>Results of the tests</th>
<th>Real optimum</th>
</tr>
</thead>
<tbody>
<tr>
<td>mt06</td>
<td>100</td>
<td>200</td>
<td>0.9</td>
<td>0.01</td>
<td>55</td>
<td>55</td>
</tr>
<tr>
<td>mt10</td>
<td>1000</td>
<td>200</td>
<td>0.9</td>
<td>0.01</td>
<td>930</td>
<td>930</td>
</tr>
<tr>
<td>mt20</td>
<td>2000</td>
<td>200</td>
<td>0.9</td>
<td>0.01</td>
<td>1170</td>
<td>1165</td>
</tr>
</tbody>
</table>

Among the methods applying GA for the JSP, the method proposed by Yamada [13] has been seen to be one of the methods which produces the best results. Therefore, we compare the results of our method with the results of Yamada’s method on the speed of convergence as well as the speed of computation. The set of parameters and the empirical results for the method Yamada proposed are shown in Table 6. In the method GA/GT of Yamada, the optimal result of the mt06 problem was found very fast after some times of test. But for the mt10 problem, the method GA/GT found out the real optimum 4 times after 600 times of test.

Table 6. The empirical results of Yamada’s method

<table>
<thead>
<tr>
<th>Problem</th>
<th>Solution size</th>
<th>Number of generations</th>
<th>pc</th>
<th>pm</th>
<th>Results of the tests</th>
<th>Real optimum</th>
</tr>
</thead>
<tbody>
<tr>
<td>mt06</td>
<td>100</td>
<td>200</td>
<td>0.9</td>
<td>0.01</td>
<td>55</td>
<td>55</td>
</tr>
<tr>
<td>mt10</td>
<td>1000</td>
<td>200</td>
<td>0.9</td>
<td>0.01</td>
<td>930</td>
<td>930</td>
</tr>
<tr>
<td>mt20</td>
<td>2000</td>
<td>200</td>
<td>0.9</td>
<td>0.01</td>
<td>1170</td>
<td>1165</td>
</tr>
</tbody>
</table>

The parallel genetic algorithm we proposed is not only better than Yamada’s method in the speed of convergence but also in reducing of the running time. This is because there are many CPUs to run concurrently for a while and send the optimal results to the server computer. The best run of the hybrid parallel genetic algorithm with 6 processors can save 3 to 4 times of running time compare to the sequential algorithm with 5 times of run on one computer. Table 7 compares the running time between the parallel genetic algorithm and the sequential algorithm with the data given in Table 5.

Table 7. Compare the running time between parallel GA and sequential GA

<table>
<thead>
<tr>
<th>Pro.</th>
<th>Sequential GA</th>
<th>Parallel GA</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Size</td>
<td>No. of run</td>
</tr>
<tr>
<td>mt06</td>
<td>100</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Since the comparison of the empirical results, it finds out that the method we proposed has the following advantages:

1. Improvement of the speed of convergence.
2. Saving of the running time and ability to run on a lower configuration computer system.
3. Easy to program because of using natural coding.

V. CONCLUSION

In this article, we apply the parallel genetic algorithm for the Job Shop Scheduling Problem. Our program was run with the input data getting from the benchmark problems which proposed by Muth and Thompson and produced a good result. The empirical results show that the parallel genetic algorithm is a very good candidate for the JSP.

REFERENCES