Solving Job Shop Scheduling Problem with Ant Colony Optimization

Cansin Turguner∗ and Ozgur Koray Sahingoz†
Computer Engineering Department
Turkish Air Force Academy
Yesilyurt, Istanbul 34196
E-mail:∗ cturguner@hho.edu.tr †sahingoz@hho.edu.tr

Abstract—Job Shop Scheduling Problem (JSSP) is one of the important and tough problem in real world, which tries to schedule N jobs to be performed on M machines. In this paper, it is aimed to imply this problem in computer science and also bringing up many related solution models. Although, most of these models try to find an optimal approach, Ant Colony Optimization (ACO) constitutes the optimal solution by using some of the complex ACO scenarios effectively. In this study, it is focused on one of the scenario of the JSSP and explained in which steps are used in the ACO solution. With simulation study, it is aimed to show, how ACO copes with the JSSP clearly.

Index Terms—Job Shop Scheduling Problem; Ant Colony Optimization; Related Solutions.

I. INTRODUCTION

The Job Shop Scheduling Problem (JSSP) is one of the well-known non-polynomial problem in both production management and combinatorial optimization areas [1][2]. Basically, in JSSP there is a work location and in which a number of general purpose work stations exist and are used to perform a variety of jobs. The main purpose is to schedule the jobs between machines efficiently. To construct an efficient solution, it is not allowed to process the same job on different machine and each job must be processed on each machine exactly once.

In recent years many algorithms have been proposed for solving this problem, such as neutral network algorithm (NNA), Genetic Algorithm, Simulated Annealing, PArtricle Swarm Optimization, Ant Colony Optimization, etc. From these solution approach, ACO has proved its success by its own peculiarity of robustness and generality [3][4]. ACO solution on JSSP is getting the solution in an acceptable time. However, when the number of jobs and machines are increased to the stochastic values and the criterion becomes more than one, ACO solution can be useless. To solve the large scale problem of jobs and machines in JSSP, multi-threading techniques can be implemented by means of agent technology. In some research these types of problems are solved with parallel implementation on different platform such as CUDA and/or multicore [5], [6], [7].

The rest of the paper is organized as follows: the second section is the overview of the job shop scheduling problem. Then, it is pointed out some related solutions on JSSP including hybrid and agent approaches in section three. In the fourth section, the proposed technique, using ACO algorithms is explained deeply. Finally, conclusions and future works are drawn.

II. JOB SHOP SCHEDULING PROBLEM DEFINITION

Job Shop is an area that many general purpose work stations are located in it. These work stations perform variety of jobs. There are four factors to describe JSSP. These are Arrival Patterns, Number of Work Stations (machines), Work Sequence, and Performance on Evaluation Criterion. There are two types of arrival patterns, static and dynamic. In static, stochastic can be occurred, complex jobs coming uncertain time periods. The important and core of the JSSP is the works sequence factor. It can be flow shop (repeated) or random (all paths are possible). This means, the jobs migrate from one machine to another that have to be passed over all machines, have to carry out a feasible schedule and have to be optimal due to its performance evaluation criterion [9]. The performance evaluation criterion involves many functions. These are makespan, average time of jobs in shop, lateness, average number of jobs in job shop and utilization of machines and workers. Although these objective functions can be considered in JSSP, makespan is the principle for researchers and is able to mention main computational difficulty to determining the optimal schedule [10]. In this paper, minimize the makespan is one of the objectives of this study. There are many scenarios of JSSP shown below.

- N jobs, 1 machine
- N jobs, 2 machines (flow shop)
- N jobs, 2 machines (any order)
- N jobs, 3 machines (flow shop)
- N jobs, M machines

In this paper, it is aimed to solve N jobs, M machines any order problem. The Constraints are shown below.

- Each jobs are processed on only one machine at a time
- Machines are available all the time
- Jobs are not depended to each other
• Setup time is included in processing time for each job.
• Any processes cannot be preempted on the machines.
• There is no priority between the jobs.

Variables, makespan and total flow time are formulated as below:
• N number of jobs.
• M number of machines.
• T is processing time and shown like T_{ij}, i is the id of job (i=1,2,3...N), and j is the id of machine (J=1,2,3...M) where the i job works on.
• S is the set of the scheduled jobs (permutation of job set).
• C is the completion time of i job on the j-th machine in time of i is appended, shown as C(S;i,j)

\begin{align*}
C(S_i, 1) &= t(S_i, 1) \\
C(S_i, j) &= C(S_{i-1}, j) + t(S_i, j) \quad j=2,...,m \\
C(S_i, j) &= \max_C(S_{i-1}, j) + T(S_i, j) \quad j=2,...,m
\end{align*}

• Makespan is defined as \( F_1 = C(S_N, M) \), i=1 to N.

III. RELATED WORKS FOR THE JSSP

If the JSSP is less complex, Simple Graphical Display Technique (Gantt Chart) is more suitable for presenting the problem without any rules and displaying and evaluating criterion results. Otherwise, the more effective techniques like GA, ACO or hybrid usages are suitable for covering the complex JSSP. For giving an example using N=5 jobs, 2 machines flow shop scenario, first job is processed in machine 1 first and then machine 2 without changing order as shown processing times in Table 1.

<table>
<thead>
<tr>
<th>Jobs</th>
<th>Machine 1</th>
<th>Machine 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>13</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>7</td>
</tr>
</tbody>
</table>

In Figure 1, two machines process the 5 flow arrival jobs on Gantt Chart. Makespan time is 38. Machine 1 has no idle time except 3 units at end of day, machine 2 has 3 units of idle time plus 1 unit at beginning of day. Jobs 2, 4 and 5 wait a total of 6 units at machine 2.

A. Genetic Algorithm Solution for JSSP

Genetic Algorithm is a stochastic optimization based on Darwinian natural selection. It has generating and testings. Genetic algorithm is an effective method in big and complex search space, there is no problem expression by mathematical modelling and no results required by existing optimization methods [11]. The steps of GA is shown Figure 2.

• The first step is to generate genetic representation. It is important to obtain suitable one. As choosing appropriate initial code will affect all the other steps [1]. According to which JSSP scenario is determined, feasible number of machines and jobs have to be handled and the chromosomes are formed.
• There is also fitness function helping to improve the optimal schedule.
• In selection step, the best selection method is selected due to the result of fitness results. Then using this selection method, two chromosome are selected to insert them into crossover section.
• After crossover, if it is not able to obtain a new generation or generation crossover is gone into deadlock, mutation techniques are applied to the generation.
• Finally, population of chromosomes are checked and analyzing the result whatever the result is feasible or not. If it is not feasible, the steps are repeated at the beginning of the compute Fitness.
B. An Agent Based GA Approach on JSSP

In this method, researchers use Genetic Algorithms to solve JSSP with multi-agent technology. One of the important reasons for using multi-agent systems is to gain multi-thread capability. Multi-threading makes parallel and distributed models for the JSSP [12]. The main propose of using agents is to getting initial population and parallelize the genetic algorithm for solving JSSP. In this architecture, choosing parallellization instead of serial structure improves the quality of genetic algorithm solution. To compare with serial GA and Agent based GA solutions, obtaining reasonable generation with agent based GA more rapid and the number of various generations have more shorter lengths than serial GA [12]. In this study, it is proposed to minimize makespan of the JSSP with ant colony optimization by multi-agents.

C. Ant Colony Optimization Approach on Problem

The ACO is the implementation of communication and exchange information mechanism of ants for searching the optimal and the shortest way to food. The mechanism works like this. Once, a pioneer ant travel from nest to its target food. When he travels on his path, he releases chemical material called pheromone inverse proportional to the length of the path [13]. Then all the other ants follow this material to arrive the food at an optimal time and also they release the pheromone. The concentration of pheromone is the main factor. And the ants can choose the optimal path due to the concentration of pheromone. It means, if there is an obstacle occurred on the well-known path (high pheromone); one can change the path to the food. And the others encounter the obstacle this like goes on. And finally, the concentration of pheromone on the path will be changed. Like shown Figure 3, the new path that has high concentration of pheromone is occurred. Pheromone rates are decreased progressively by evaporation.

This approach is also heuristic and pheromone rates that implies the real world of ants [14]. The ACO algorithm works like this;

- Initialize data
- Construct ant solutions
- Apply local search
- Update pheromones table
- Repeat the steps until reached the optimal makespan or maximum iteration.

To initialize the data, the JSSP parameters are exposed like set of jobs, number of nodes, etc. The pheromone values table is set up by utility function that gives the feasible solution after certain irritation in the minimum makespan. Centralized actions are performed by not only one ant [15]. This is application of local search. Before decision of which pheromone value is changed, local optimization have to be applied to the solution. Due to the solution, the pheromone value is increased if the solution is good, else the value of pheromone value is decreased. All the ants trail all the nodes in the environment due to heuristic and pheromone values. The paths that are chosen by ants are appeared in the permutation process [14] [16]. Selection of the paths are according to the heuristic and pheromone factors. In this paper, heuristic value is indicated as $H_{ij}$ and pheromone value is indicated as $P_{ij}$. Heuristic value is formed by the pheromone value of the ants that have found the feasible solution. To Formalize ACO as a solution for JSSP:

The utility function (Trade-off between heuristic and pheromone factors) that is determined shown in Equation 1 and Equation 2.

$$F(utility)_{ij} = \frac{H_{ij} \cdot P_{ij}}{\sum_{i,j=1}^{n,m} H_{ij} \cdot P_{ij}}$$

$$H_{ij} = \frac{1}{F(X_j)}$$

Heuristic value is computed from $F(X)$ function that $X$ is defined as a cost. Before modifying the pheromone value, local search rule is applied.

In simple ant colony optimization (S-ACO), there are two modes of ants. They are forward and backwards. Owing to the step by step solution of ACO, ants choose and trace a path to the food as a forward move. And then back-trace the path to the nest as a backward move. Depositing pheromones is being occurred only in backward mode.

In solution construction of JSSP, every ants have the capabilities of realizing and smelling (indirect communication) pheromones. And ants choosing one of the jobs probability is relative weight of the pheromone trail of the jobs. Example formula of probability is shown in Equation 3.

$$P_{ij}^k = \frac{[\tau_{ij}]^\alpha \cdot [\eta_{ij}]^\beta}{\sum_{t \in N_i^j} [\tau_{it}]^\alpha \cdot [\eta_{it}]^\beta}$$

After that, a solution is set by an ant and the pheromone table is updated according to this new values. As a result next ants can perform better selections. Pheromone values are evaporated before starting new tours. $\rho$ is defined as the evaporation rate and it is calculated according to Equation 4.

$$\tau_{ij} \leftarrow (1 - \rho) \cdot \tau_{ij}$$
Each moving ant leaves some additional pheromones and therefore pheromone values must be updated according to Equation 5 while ants completing their path constructions.

$$\tau_{ij} \leftarrow \tau_{ij} + \sum_{k=1}^{m} \Delta \tau_{ij}^k$$  (5)

IV. **Proposed Solution: Solving Job Shop Scheduling Problem with ACO**

In this project it is coped with NXM flow order type JSSP. The number of the artificial ants is N. It means the number of the ants is equal to the number of the jobs. ACO algorithms are iterative algorithms that need many iteration to succeed the problem. Many iterations cause many CPU times. Although to lower the time, increasing the number of the ants are appropriate but the number of the ants is a big problem as known all of the ants are also threads so this approach causes overweight for the system resources especially for the CPU. However, decreasing the process time is relatively to the number of ants , determination of how many ants can be useful for the system is crucial. It is aimed to reach global optimum using many iterations in less time. This approach is like a parallel solution. In theoretical, the number of ants have to be equal to the jobs of the JSSP according to this study. But practically it must be limited. To define the proposed model clearly, the problem have to be minimized. So in this proposed model, the data set is shown in Table 2.

**TABLE II**

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Job 1</td>
<td>10</td>
<td>8</td>
<td>7</td>
<td>14</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>9</td>
</tr>
<tr>
<td>Job 2</td>
<td>5</td>
<td>6</td>
<td>10</td>
<td>9</td>
<td>12</td>
<td>5</td>
<td>3</td>
<td>5</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>Job 3</td>
<td>8</td>
<td>6</td>
<td>5</td>
<td>1</td>
<td>4</td>
<td>4</td>
<td>2</td>
<td>4</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>Job 4</td>
<td>3</td>
<td>12</td>
<td>2</td>
<td>7</td>
<td>4</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>Job 5</td>
<td>6</td>
<td>8</td>
<td>4</td>
<td>1</td>
<td>8</td>
<td>2</td>
<td>4</td>
<td>2</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>Job 6</td>
<td>10</td>
<td>4</td>
<td>6</td>
<td>10</td>
<td>2</td>
<td>1</td>
<td>7</td>
<td>3</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>Job 7</td>
<td>23</td>
<td>5</td>
<td>7</td>
<td>8</td>
<td>2</td>
<td>4</td>
<td>4</td>
<td>6</td>
<td>10</td>
<td>12</td>
</tr>
<tr>
<td>Job 8</td>
<td>6</td>
<td>2</td>
<td>7</td>
<td>3</td>
<td>7</td>
<td>8</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>Job 9</td>
<td>9</td>
<td>3</td>
<td>5</td>
<td>5</td>
<td>2</td>
<td>4</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>Job 10</td>
<td>1</td>
<td>9</td>
<td>3</td>
<td>12</td>
<td>2</td>
<td>5</td>
<td>4</td>
<td>12</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>

Constraints are:

- process capabilities of machines are limitless.
- The process times of each jobs for each machines are defined before by the developer.
- Both number of iteration and global optimum values are defined as an ending criteria by the developer.
- Pheromone trail limitations.

The proposed method seems like an example of the MIN-MAX Ant System. But it has some differences. One of the difference is; In this project, it is used only the iteration best approach means, not both iteration best and total makespan best. The disadvantages of this approach is the solution can be caught in local optimums. The save the solution from local optimums, maximum and minimum pheromone trails for the limitation. And it is used same constant value for increasing and evaporating the pheromone trails. Flow chart of the proposed model is shown in Figure 4.

![Flow Chart for the Proposed Model](image)

Initializing the parameters is the first step. Jobs and their process times for each machines are defined. Then minimum makespan times of the jobs are calculated. The jobs that has best results are distributed to the ants. In this study, in determination of the fist jobs of the ants, randomization is not used. Because using the best jobs instead of random jobs lowers the number of the iterations at the beginning.

In pheromone table, values are the same for every jobs when it is creating. After iterations this can be changed. Also after iterations which ant do the best iteration only has the authority for update the pheromone table.

Distribution of the jobs are up to the probability array. Every ants use this array when selecting the job. Also the ants never get the same job that has before in the same iteration.

Fitness control for local optimum to obtain and designate best iteration solution. If it is the best, then the ant that does this solution updates the pheromone table. Max and minimum pheromone rates are used at this point. If the current pheromone value becomes upper than the max, maximum
value is set. If the current pheromone value becomes lower than the min, then minimum value is set.

Finally, ending criteria can be the number of the iteration or definite makespan time. In this method, it is used definite makespan time to show how this study work properly and how this study result is better than the other studies results. Simulation results for the proposed model is shown in Table 3.

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>150</td>
<td>142</td>
<td>149</td>
<td>148</td>
<td>151</td>
<td>157</td>
<td>160</td>
<td>165</td>
<td>153</td>
<td>143</td>
</tr>
<tr>
<td>152</td>
<td>150</td>
<td>143</td>
<td>151</td>
<td>149</td>
<td>155</td>
<td>157</td>
<td>164</td>
<td>150</td>
<td>140</td>
</tr>
<tr>
<td>149</td>
<td>141</td>
<td>152</td>
<td>146</td>
<td>143</td>
<td>152</td>
<td>143</td>
<td>163</td>
<td>148</td>
<td>138</td>
</tr>
<tr>
<td>145</td>
<td>140</td>
<td>156</td>
<td>141</td>
<td>156</td>
<td>150</td>
<td>134</td>
<td>161</td>
<td>145</td>
<td>141</td>
</tr>
<tr>
<td>143</td>
<td>138</td>
<td>146</td>
<td>145</td>
<td>143</td>
<td>149</td>
<td>134</td>
<td>158</td>
<td>145</td>
<td>137</td>
</tr>
<tr>
<td>150</td>
<td>150</td>
<td>143</td>
<td>145</td>
<td>144</td>
<td>146</td>
<td>140</td>
<td>158</td>
<td>145</td>
<td>133</td>
</tr>
<tr>
<td>141</td>
<td>140</td>
<td>145</td>
<td>136</td>
<td>140</td>
<td>149</td>
<td>132</td>
<td>154</td>
<td>141</td>
<td>132</td>
</tr>
<tr>
<td>141</td>
<td>133</td>
<td>152</td>
<td>141</td>
<td>133</td>
<td>145</td>
<td>133</td>
<td>152</td>
<td>140</td>
<td>131</td>
</tr>
<tr>
<td>130</td>
<td>132</td>
<td>145</td>
<td>137</td>
<td>131</td>
<td>144</td>
<td>132</td>
<td>150</td>
<td>139</td>
<td>131</td>
</tr>
</tbody>
</table>

V. CONCLUSION

In this study, we encounter with many problems when developing ACO. The most clear one is the CPU usage rate. The other one is sometimes local optimum can be occurred instead of global optimum. To cope with this problem, static max and min value of pheromone values are used. The experimental results shows that ACO produces an acceptable solution for JSSP.

In the future work, the simulation of the proposed model will be occurred. How efficient using multiagent based structure with this proposed model will measured. Also after developing this structure, it is calculated how the number of iterations and the time for processing will be decreased.

REFERENCES
