An Improved Ant Colony Optimization Algorithm for Job Shop Scheduling

Sayyad Gulammustafa G and Kolekar Vijay Sudam
Department of Computer Engineering, DGOIFOE, Swami-Chincholi, Daund – 413130, India
gmustfa22@gmail.com, vijay.kolekar20@gmail.com

Abstract - The problem of efficiently scheduling jobs on several machines is an important consideration when using Job Shop scheduling production system (JSSP). JSSP is known to be a NP-hard problem and hence methods that focus on producing an exact solution can prove insufficient in finding an optimal resolution to JSSP. Hence, in such cases, heuristic methods can be employed to find a good solution within reasonable time. In this paper, we study the conventional ACO algorithm and propose a Load Balancing ACO algorithm for JSSP. We also present the observed results, and discuss them with reference to the conventional ACO. It is observed that the proposed algorithm gives better results when compared to conventional ACO.

Keywords - Ant Colony Optimization, job shop scheduling, task scheduling, load balancing, ACO

I. INTRODUCTION

The classical Job Shop Scheduling Problem (JSSP) considers the problem of efficiently scheduling a finite number of jobs to a finite number of machines for processing. Each job consists of a sequence of operations which have to be processed using a specified machine for a specified amount of time, without any interruption. The operations belonging to the same job have a technological sequence and none of them should begin processing before the preceding operation has finished its execution. The challenge is to find a feasible schedule consisting of the assignment of operations on machines without violating these constraints. Also, the solution must specify the optimum makespan for the schedule. The makespan is defined as the maximum completion time of all the jobs considered. To optimize the makespan, it is necessary to make sure that the idleness of machines is minimized. Thus, JSSP is a NP-hard combinatorial optimization problem and obtaining the actual solution for JSSP is computationally difficult.

In this paper, we present an analysis of the application of the metaheuristic Ant Colony Optimization technique for the job shop scheduling problem. Ant Colony Optimization (ACO) technique is inspired by foraging behaviour of ants in nature. ACO tries to mimic the observed behaviour of ants while they conduct a search for an efficient path to follow to carry their food back to the nest. In a similar fashion, in ACO, the concept of an ant is considered. Each ant constructively builds a solution to the problem at hand by making decisions using path probabilities at each decision point.

ACO has been used extensively to present effective solutions to many combinatorial optimization problems like Travelling Salesman problem and Vehicle routing problem. In this paper, we apply and analyze the effectiveness of Ant colony optimization for JSSP. Furthermore, the results of ACO are improved by adding a load balancing factor while calculating probabilities. It was observed that, the inclusion of load balancing factor improves the results drastically and the proposed algorithm outperforms conventional ACO.

The paper is organized as follows – Section 2 presents related work done on this topic. Section 3 sheds more light on the intricacies of the Job Shop scheduling problem. Section 4 describes the Ant Colony Optimization and its procedure as applied to JSSP. Section 5 presents a detailed discussion on the proposed load balancing ACO.
Section 6 presents the results obtained and their analysis followed by conclusion and references.

II. RELATED WORK

Job Shop scheduling problem and ACO has been a problem of constant research and has attracted the attention of several researchers. In recent years, researchers have proved that ACO performs well as compared to other metaheuristic approaches when applied to optimization problems. The first Ant algorithm was proposed by Dorigo et al [6] in 1992 as a way to solve the Travelling Salesman Problem (TSP). They suggested ACO based approaches to solve some of the complex quadratic assignment problems. When it comes to the field of scheduling, ACO has been successfully applied to solve challenging problems like vehicle routing, graph coloring, sequential routing, etc. Also, den Besten and Juan [9] applied ACO for single machine weighted tardiness problem which is also a NP-hard scheduling problem and consists of the problem of efficiently scheduling a given set of jobs on a single machine. Omar and Baharum [10] successfully solved JSSP using a Genetic Algorithm based approach.

We conducted a study on how ACO can be applied to the problem of scheduling. It was found that ACO can be used to compete with the results of other metaheuristic approaches like Genetic algorithm and Bee intelligence. Zhang and Hu [1] and Sun et al. [2] applied the concept of ACO to JSSP. They calculated the probability of finding next path using two parameters: task length and pheromone intensity. Their algorithm gave an optimal solution for small JSSP problems. But, as the problem size increased, the deviation from optimal solution became more significant. Most of the ACO based algorithms proposed previously do not take the machine load into consideration. The proposed algorithm uses this factor of machine load and is capable of performing better than conventional ACO.

A. Ant Colony Optimization(ACO)

An Ant system is based on the interesting foraging behaviour of ants observed in nature. [6] Ants are capable of finding the shortest path from food to their nest, without using visual cues. They actually use a hormone called pheromone deposited by other ants and its concentration levels to decide the best path. While walking, ants deposit pheromone on the path and the ants coming later choose the path with a greater concentration of the pheromone, since this indicates the way to the nest. The technique based on this idea is called Ant Colony Optimization.

The basic idea in ACO is to use a population of ants to iteratively build a solution by continually applying a decision policy based on probability until a solution is found. Ants that find a good solution mark their paths with pheromones. So, in the next iteration, ants are attracted towards to the pheromone which results in greater chances of following good paths. ACO has a memory which stores the components of the path being traversed. So, at the end of any iteration we can get the path used by the ants.

In-order to apply ACO algorithm, the optimization problem must be plotted in a graphical representation G. The pheromone level at each edge is initialized to a positive real value c. Each ant is positioned at the starting node. The ant then starts traversing the graph using the values obtained from probabilistic computations, until it reaches the destination node. The path used by the ant is recorded and the best solution will be recorded. The pheromone amount of the path used by ant is then determined and another ant will start its traversal if the stopping criteria is not met. [1]

The ant uses a probabilistic computation to determine the next path during its traversal. The computation is based on two parameters: the pheromone present along the edge and the weight of the edge. The probability to move from node i to node j for each kth ant at time t can be defined as:

\[ P(i,j) = \begin{cases} \frac{\tau(i,j)^\alpha \eta(i,j)^\beta}{\sum_{k \in S_k} \tau(k,u)^\alpha \eta(k,u)^\beta}, & \text{if } j \in S_k \\ 0, & \text{otherwise} \end{cases} \]  

where \( \tau(i,j) \) represents the pheromone trail, \( \eta(i,j) \) represents the edge weight. The
parameters $\alpha$ and $\beta$ determine the degree to which pheromone level is used as against the weight of the edge. The values of $\alpha$ and $\beta$ are adjusted such that the node having the edge with less weight and higher pheromone levels is selected.

After selecting the path, the ants will lay the pheromone according to equation 2.

$$\tau(i,j) = (1 - \rho) \cdot \tau(i,j) + \rho \cdot \tau_0$$

(2)

where $\rho$ is the real valued coefficient such that $(1-\rho)$ becomes the evaporation coefficient of the edge $\text{(i,j)}$. The value of $\rho$ must be between 0 and 1 i.e. $\rho \in (0,1)$. The pheromone deposited by $m$ ants is determined by:

$$\tau(i,j) = (1 - \rho) \cdot \tau(i,j) + \rho \cdot \Delta \tau(i,j)$$

(3)

where

$$\Delta \tau(i,j) = \begin{cases} L_{gb} & \text{if } (i,j) \text{global best tour} \\ 0 & \text{otherwise} \end{cases}$$

The ant system has an important property of pheromone evaporation which causes the pheromone deposited to decrease over the period of time. This property helps in preventing premature convergence to a sub-optimal solution.[1][4]

### B. Ant Colony Optimization applied to JSSP

To apply ACO for JSSP, it is necessary to plot the given JSSP problem in the form of a graph where nodes represent the operations and the execution time represents the edge weights.[1] Two dummy nodes, Start and Destination are added to the graph. Once the graph is plotted, the problem converts to Travelling Salesman Problem and we have to find the optimal path from Start to Destination.

Finding the optimal path to be followed is carried out as follows - Ants are placed at the Start node, and are made to traverse the whole graph and reach the Destination node. While traversing, each ant is allowed to visit a node only once. Whenever the ant wants to move to next node, it calculates the probability of other nodes. But, unlike TSP, in JSSP, probability for each remaining node is not calculated since technological sequence needs to be satisfied. So, to satisfy the constraint, the visiting set $S_k$ is maintained. $S_k$ is initialized to starting nodes of each operation i.e. $S_k = \{u_{i1} \mid i \in \{1, n\}\}$ where $n$ is number of jobs. The ant is allowed to choose the next node only from those in $S_k$. The ant uses formula (1) to calculate the probability of selecting the next path. After an ant chooses the nodes in $S_k$ to visit, the chosen node is removed and its successor node in the given task is added in its place. The procedure is continued until all nodes are visited and the ant reaches the destination node. After the ant reaches the destination node, its path cost is calculated. Figure 2 depicts an example path used by ant for moving from Start to Destination node.

![Figure 2: Path used by ant](image)

Figure 3 shows the procedure followed while applying Ant Colony Optimization to JSSP.

```
begin
  Initialize the parameters
  While (Stopping criterion not met)
    Position each ant at Start node
    while(Stop each ant has build a solution)
      for each ant do
        Choose next node by pheromone trail & edge weight
        end for
      end while
    Update the pheromone
  end while
end
```

Figure 1: Plotting 3x3 JSSP into graph
III. MATERIALS AND METHODS

The conventional ACO considers pheromone trails and the edge weights while calculating the probability which is used for choosing the next operation. There is a need to consider machine load as a decisive factor while calculating probability. The proposed Load balancing ACO adds another parameter - Machine load.

A. Choosing the next operation

The conventional ACO algorithm does not focus on the machine load factor. However, to minimize idle time, load balancing of machines is an important factor. If a particular machine corresponding to an operation is heavily loaded, the probability of completion for that operation is reduced and other operations whose machines are less loaded are preferred. Thus, in the proposed Load balancing ACO, the ant chooses the next node to be visited using the formula:

\[
P(i,j) = \begin{cases} 
\frac{\tau(i,j)^{\alpha} \eta(i,j)^{\beta} \lambda(i,j)^{\gamma}}{\sum_{u \in S_k} \tau(i,u)^{\alpha} \eta(i,u)^{\beta} \lambda(i,u)^{\gamma}}, & \text{if } j \in S_k \\
0, & \text{otherwise} 
\end{cases}
\]

where \(\lambda(i,j)\) represents the total execution time of the machine corresponding to the operation. Whenever a task is allotted to a particular machine, the task execution time is added to total execution time of that machine. An array is used to store the total execution time for each machine.

If a machine is already executing an operation, choosing the next operation to be executed on the same machine increases idle times of other machines. Hence, with a view to decrease the idle time, the machine load parameter is considered. Due to this, machines with lower execution time are preferred resulting in overall reduction of idle time.

B. Calculating the cost

While calculating the cost, nodes are distinguished in two parts as: Initial node and Other nodes. Initial node is the starting node for any job. For example, if a job consists of operations = \{1, 2, 3, 4\}, then node 1 is the initial node and nodes 2, 3 and 4 are the other nodes. Also, the completion time (the time at which node completes its execution) of each node is stored in the memory.

\[
\text{Cost} = \text{maximum of all machine times}
\]

The above process is repeated for each ant in each of the iterations. In the above flowchart, machine time means the total execution time of machine corresponding to the node. For each ant, its path cost is calculated and if the cost is smallest, it is stored for further comparison. At the end of all iterations, this cost is given as the output.

IV. TEST RESULTS

A. Visualization of JSSP

In our work, we developed a technique to show the graphical representation of the JSSP problem and the path chosen by the ant for 3x3, 6x6 and 12x12 JSSP using the proposed Load Balancing ACO algorithm. These are shown in the figures 5, 6 and 7. In each figure, the first part shows the initial graph setup i.e. the red dots (nodes) represent the tasks to be executed on machines. The second part shows the path used by the ant to traverse from Start to Destination node. The edges in the graph represent the path.
used by the ant. The cost of this path is calculated and displayed in the text area. The values in the text area below each graph represent the best solution and average solution obtained in time units. The first value indicates the best solution while the second one represents the average solution produced as a result of all iterations.

Figure 5 shows the path used by the ant to traverse from Start to Destination node for 3x3 JSSP. Here, 15 time units is obtained as the best solution and 15.7 time units being the average solution. Figure 6 shows the path used by ant while traversing from Start to Destination node for 6x6 JSSP. The best solution obtained is 55 time units and 83.6 time units is the average solution. Figure 7 depicts the path traversed by ant while solving JSSP of size 12x12. The best solution obtained as a result is 1322 time units while 1806.4 is the average solution.

All the above examples represent problems of format n x n. The proposed algorithm can also applied to JSSP of size in the format n x m where n ≠ m.

### B. Experimental Evaluation

The main aim of the work presented here was to use ACO for the Job Shop Scheduling problem and analyze the improvement in performance due to the inclusion of a load balancing factor. The experiments were carried out using ACO and Load balancing ACO algorithms. The OR-library [8] which is a collection of test datasets for a variety of Operations Research problems provides a collection of benchmarks for Job Shop Scheduling. These problems are numbered in format “ftn” where n represents number of jobs and m represents the machines used for the problem. In these benchmark problems, n = m; hence we have used the same format in order to be able to present a comparative evaluation. However, the proposed algorithm can be applied to n ≠ m JSSP problems also.

The benchmark problems from OR-library: ft03, ft06 and ft10 were used [8] with known optimal values of 15, 55 and 930 units. In these experiments, the values of parameters are very important. With the experiments carried out, it has been seen that α=0.1, β=-2.0, ρ=0.01 and γ=-9.0 gave optimal values. The value of γ played an important role while achieving optimal solution.

From the figure 8, it can be seen that the makespan reduced with the increase in the number of iterations. But, after 100 iterations the
change became slower and after 500 iterations the best solution was found in most cases. So, for these experiments 500 iterations were used. Also, Load Balancing ACO requires lesser iterations for finding optimal solution as compared to ACO. At the 25th iteration, ACO found 1796 time units to be the solution while Load Balancing ACO found out 1441 time units as the solution.

Figure 8: Variation of makespan with iterations for 12 x 12 JSSP

V. DISCUSSION ON PROPOSED SYSTEM AND RESULTS

Figure 9 shows the performance improvement of proposed Load Balancing ACO over conventional ACO. Table 1 shows the results obtained after the experiments. It indicates the optimal solution (makespan times) given by both the algorithms. From the graph in Figure 9 and Table 1, it can be observed that conventional ACO obtained near-optimal solution for smaller complexities. But, as the complexity increased, the deviation from optimal solution increased significantly. Hence, the Load Balancing ACO has been proposed which takes into account the Machine load factor. Figure 9 shows that the proposed Load Balancing ACO showed better performance for each problem. For the 6x6 JSSP, the improvement observed was a mere 2.7%, while for 10x10, it was 9.3%. This increased to 17% for the 12x12 Job Shop Scheduling problem. Load Balancing ACO helped in reducing deviation from optimal solution and hence showed better results as compared to conventional ACO.

Thus, experimental results showed that our proposed algorithm outperforms conventional ACO. This performance improvement increases as the complexity of the JSSP grows. Thus, addition of load balancing factor helped in assigning equal load to machines and the effective makespan time decreased.

![Figure 9: Comparison of load balancing ACO over conventional ACO](image-url)

<table>
<thead>
<tr>
<th>JSSP Name</th>
<th>(n) (No.of Jobs)</th>
<th>(m) (No.of machines)</th>
<th>Observed Makespan Times</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Traditional ACO (best case)</td>
</tr>
<tr>
<td>ft03</td>
<td>3</td>
<td>3</td>
<td>15</td>
</tr>
<tr>
<td>ft06</td>
<td>6</td>
<td>6</td>
<td>56.5</td>
</tr>
<tr>
<td>ft10</td>
<td>10</td>
<td>10</td>
<td>1219</td>
</tr>
<tr>
<td>12x12</td>
<td>12</td>
<td>12</td>
<td>1590</td>
</tr>
</tbody>
</table>

TABLE 1. COMPARISON OF MAKESPAN OF LOAD BALANCING ACO WITH CONVENTIONAL ACO FOR JSSP

VI. CONCLUSION AND FUTURE WORK

In this paper, we proposed a Load Balancing ACO algorithm for achieving better results in Job Shop scheduling problem. We have also
evaluated and compared the results obtained with conventional ACO algorithm and found that our proposed algorithm gives better performance. As the size of JSSP increases, the difference in the results produced by both these algorithms increases and Load Balancing ACO proved to be more useful for JSSP problems of higher complexities.

As observed during use, Load Balancing ACO showed great performance for JSSP. As part of future work, we are currently working on applying the proposed algorithm to a cloud based environment to observe the results of a load balancing scheduling algorithm in a virtualized environment.

REFERENCES


