CHAPTER 6

HYBRID ANT COLONY OPTIMIZATION FOR GRID SCHEDULING

6.1 INTRODUCTION

Heuristic algorithms are algorithms which find best solutions among possible solutions Askarzadeh & Rezazadeh (2013) and Bagloee et al (2013). Heuristic algorithms do not guarantee finding best solutions. They are fast and are classified into Divide and Conquer, Branch and Bound, Dynamic Programming and Local Search. For NP complete problems, Branch and Bound and Dynamic Programming techniques are not preferred due to high time complexity.

Metaheuristic algorithms are heuristic algorithms to find a lower-level heuristic that ensures a good solution for an optimization problem. Swarm Intelligence Parpinelli & Lopes (2011) and Evolutionary Algorithms (Derrac et al 2011) are popular Metaheuristic algorithms. The collective decentralized behavior of natural/artificial phenomena is the basis for Swarm Intelligence. Evolutionary algorithms are inspired by biological evolution like reproduction, recombination and selection.

Metaheuristics like Simulated Annealing (SA), Genetic Algorithm (GA) (Holland 1975, Gkoutioudi et al 2012 & Delavar et al 2010), ACO Bonabeau et al 2000 & Liu et al 2014 (Jain & Singh 2014), Particle Swarm Optimization (PSO) (Kennedy 2010 & Izakian et al 2010) were proposed for
grid scheduling as they ensure higher quality results than simple heuristics Kashyap & Vidyarthi (2011). This work investigated the ACO algorithm for computational grid scheduling problems. An improved ACO based on a mutation concept of Differential Evolution is proposed.

Differential Evolution (DE) is a parallel, iterative method to find an optimal solution regarding a given measurement about quality. DE algorithms were used in engineering for optimization as it is fast, simple and not based on gradients. DE algorithm took a candidate solutions population called agents. The agents in search space were moved using a mathematical model and merge with previous agents locations in the population. If agent’s new positions are better than previous agent’s positions, it is taken for next iteration, otherwise discarded. Agents searching and merging is repeated till a satisfactory solution is retrieved. The algorithms steps are given in Figure 6.1.

![Figure 6.1 General Evolutionary Algorithm Procedure](image)

Policies and strategies for mutation/crossover strategy affect DE algorithms performance. Their control parameters are population size (NP), Crossover Rate (CR) and scale factor (F) (Liu & Lampinen 2002, Das & Suganathan 2011). Different strategy combinations and control parameters based on the type of problem needed optimization were selected. Selection of best strategies and parameters is a trial-and-error, high cost, time consuming
process. DE algorithm does not guarantee a solution which is its disadvantage.

Animals or insects behavior were applied in optimal search problems. Some popular algorithms are PSO, ACO algorithm and Artificial Honey Bee Algorithms. Recently ACO had a big impact on discrete optimization problems. The Travelling Salesman Problem, Network Routing and Scheduling use ACO optimization.

The hybridization level is the degree of coupling between meta-heuristics, execution sequence and control strategy (Xhafa et al 2009).

Hybridization Level: Loosely coupled – where hybridized meta-heuristics preserve their identity, namely, their flow is fully used. This case is referred to as high level hybridization. Strongly coupled - where hybridized meta-heuristics inter-change inner procedures, resulting in a low hybridization level.

An optimal schedule optimizes flowtime/makespan. The motivation to use hybrid algorithms is locating a schedule where all tasks completion time is minimal. The aim is to improve results using ACO algorithm (Radha & Sumathy 2013). Traditional optimizations are deterministic and they provide exact answers quickly but they are often stuck in local optima. So, another approach is needed as conventional methods are not applicable to modern heuristic general purpose optimization algorithms.

Heuristic based algorithms specifically, population based heuristics suit grid environments task scheduling, but complex population based heuristics take long execution times. The most popular/efficient grid scheduling meta-heuristics are ad-hoc, local search and population-based methods.
Experiments were conducted with grid nodes where DE algorithms find best set of available resources and ACO algorithm schedule resources optimally.

6.2 METHODOLOGY

Two Hybrid ACO algorithms based on Differential Evolution and Random Local Search was proposed.

Hybrid DE algorithms have shown faster and reliable convergence performance than classic DE algorithms without parameter control for many benchmark problems (Zhang & Sanderson 2009). Finding the most suitable DE variant leads to it being an underlying scheme where parameter adaptation operations were introduced. Some adaptive algorithms were developed based on a classic DE/rand/1/bin which is robust but less efficient regarding convergence rate.

**Differential Evolution algorithm**

The algorithm is as follows (Brest & Maučec 2011):

Initialization(); {Generate uniformly distributed random population}

while not termination condition met do

for (i = 0; i < NP; i++) do

Select random indexes r1, r2, and r3 to be different from each other and from the index i.
\[ v_{ij}^{(G)} = x_{ij}^{(G)} + F \times (x_{ij}^{(G)} - x_{ij}^{(G)}) \]

\[ \text{jrand} = \text{rand}\{1,D\} \]

for( j = 0; j < D; j++) do

if(\( \text{rand}(0,1) \leq \text{CR} \quad \text{or} \quad j = j_{\text{rand}} \)) then

\[ u_{i,j}^{(G)} = v_{i,j}^{(G)} \]

else

\[ u_{i,j}^{(G)} = x_{i,j}^{(G)} \]

end if

end for

if(\( f(u_{i}^{(G)}) \leq f(x_{i}^{(G)}) \)) then

\[ x_{i}^{(G+1)} = u_{i}^{(G)} \]

else

\[ x_{i}^{(G+1)} = x_{i}^{(G)} \]

end if

end for

end while

In the new approach, DE algorithms found suitable resources among available resources in a network for a grid node running application. ACO scheduled resources optimally. A DE-ACO combined approach improved applications performance significantly.
6.2.1 Random Local Search (RLS)

Local search is a combinatorial optimization technique applied to solve NP-hard optimization problems (Gu 1993). Local search refines initial solutions to find a best solution by searching its local neighborhood. Local optima is found by iteratively improving initial solution in small increments. Efficient local search heuristics like random selection, pre and partial selection were used (Sosic & Gu 1991). The main strategies in local search are random search and search direction. In a random search, search direction was randomly selected while some criteria are used to find search direction in the latter. Random Local Search (RLS) gets started with assignment of a value to every variable that iteratively improves assignment by improving steps (taking random steps or by restarting with another complete assignment).

Procedure Local-Search(V,dom,C)

Inputs

V: a set of variables

dom: a function so that dom(X) is a domain of variable X

C: set of constraints to be satisfied

Output

complete assignment that satisfies constraints

Local

A[V] an array of values indexed by V

repeat

for each variable X do

A[X] ← a random value in dom(X);
while (stopping criterion not met and A is not a satisfactory assignment)

Select a variable Y and a value V ∈ dom(Y)

Set A[Y] ← V

if (A is a satisfactory assignment) then

return A

till termination

6.2.2 Proposed Hybrid Algorithms

DE algorithm converges and slows down at initial search rapidly and so a modified version named as ACO-DE used based on the concept of opposition based learning to initialize a population, random localization to select a base vector. Besides this, the concept of ant’s movement was applied as the algorithm approaches global optimum. Also, a population set was used in contrast to a double population set as in basic DE. ACO-DE basic structure is similar to DE and the algorithm’s step by step procedure is given here:

Step 1 : Initialization: Randomly generate a set P of NP individuals, generate another set OP of size NP using opposition based method and take NP fittest individuals from union of the 2 sets as initial population S. Input scaling factor F, crossover rate Cr and σ=1.

Step 2 : Mutation: Randomly select 3 distinct individuals X_{r1}, X_{r2} and X_{r3} from population S and perform mutation and individual X_{rb} is best of the 3 individuals and X_{r2}, X_{r3} are the other two.
Step 3 : Crossover: It recombine every target vector \(X_i\) with perturbed individual that are generated in step 4 to generate a trial vector \(U_i\).

Step 4 : Selection: Calculate objective function value at a new generated individual. If it is better than target individual then target individual should be replaced by new individual in current population.

Step 5 : Refining the Global Best Solution Using Ant Movement: For all iteration an individual in neighbourhood of global best individual \(X_{\text{best}}\) is generated using ant colony algorithm.

If this, individual is better than worst individual in a population then replace the worst by a new individual.

Update \(\sigma\) after every iteration as \(\sigma = \sigma \times 0.30\) if \(\sigma < 10^{-3}\) then \(\sigma = 10^{-3}\).

Step 6 : Check whether termination criterion met. If yes, then stop; otherwise move to step 2.

Such modifications ensured that the algorithm got a better tradeoff between convergence rate and robustness. So, it is possible to increase the DE algorithm’s convergence rate and obtain an acceptable solution with lower objective function evaluations. Such improvement in evaluating a candidate solution is computationally less expensive and so it is better to locate a global optimum or a good suboptimal solution.

This study hybridized Random Local Search (RLS) with ACO ACO-RLS. The ACO algorithm provides an initial solution improved by local search by locating local optima. RLS improved an initial solution point
provided by ACO algorithm and moved it to a refined solution point. This random strategy is simple and effective.

6.3 EXPERIMENTAL RESULTS

Experiments were simulated in a grid environment with five and fifty resources and various number of jobs. Each grid node has different type of resources forming a heterogeneous environment. Makespan is a widely used scheduling performance and is applicable for batch jobs and measures scheduling performance.

The Makespan is defined as the maximum time required to complete all meta-task jobs. Makespan measures the length of the complete schedule. Figure 6.2 shows the makespan achieved for different techniques.

<table>
<thead>
<tr>
<th>Number of jobs</th>
<th>ACO</th>
<th>ACO with RS</th>
<th>Proposed ACO</th>
<th>Proposed ACO with RS</th>
<th>ACO-RLS</th>
<th>ACO-RLS with RS</th>
<th>ACO-DE</th>
<th>ACO-DE with RS</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>87.96</td>
<td>84.5</td>
<td>82.51</td>
<td>80.2</td>
<td>81.29</td>
<td>79.02</td>
<td>80.29</td>
<td>78.06</td>
</tr>
<tr>
<td>300</td>
<td>272.45</td>
<td>269.24</td>
<td>259.5</td>
<td>253.46</td>
<td>256.29</td>
<td>250.08</td>
<td>253.26</td>
<td>247.16</td>
</tr>
<tr>
<td>500</td>
<td>452.68</td>
<td>439.99</td>
<td>426.64</td>
<td>414.6</td>
<td>422.25</td>
<td>409.49</td>
<td>417.69</td>
<td>403.74</td>
</tr>
<tr>
<td>700</td>
<td>617.41</td>
<td>604.44</td>
<td>580.84</td>
<td>564.43</td>
<td>573.17</td>
<td>556.32</td>
<td>565.16</td>
<td>548.51</td>
</tr>
<tr>
<td>900</td>
<td>807.22</td>
<td>781.61</td>
<td>750.92</td>
<td>733.57</td>
<td>742.38</td>
<td>723.19</td>
<td>732.78</td>
<td>714.13</td>
</tr>
</tbody>
</table>

It is seen from Table 6.1 and Figure 6.2 that the proposed ACO-DE with RS achieved lower makespan compared to ACO-RLS with RS scheduling and Proposed ACO with RS scheduling. ACO-DE scheduling with RS achieved 1.17% to 1.41% less makespan than ACO-RLS with RS. ACO-DE scheduling with RS achieved 2.52% to 2.86% less Makespan than proposed ACO with RS scheduling.
Table 6.2 Makespan (in seconds) for 50 Resources

<table>
<thead>
<tr>
<th>Number of jobs</th>
<th>ACO</th>
<th>ACO with RS</th>
<th>Proposed ACO</th>
<th>Proposed ACO with RS</th>
<th>ACO-RLS</th>
<th>ACO-RLS with RS</th>
<th>ACO-DE</th>
<th>ACO-DE with RS</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>9.38</td>
<td>9.27</td>
<td>9.02</td>
<td>8.89</td>
<td>8.91</td>
<td>8.76</td>
<td>8.8</td>
<td>8.65</td>
</tr>
<tr>
<td>300</td>
<td>30.7</td>
<td>30</td>
<td>28.89</td>
<td>28.52</td>
<td>28.49</td>
<td>28.21</td>
<td>28.14</td>
<td>27.93</td>
</tr>
<tr>
<td>500</td>
<td>48</td>
<td>47.4</td>
<td>45.94</td>
<td>45.3</td>
<td>45.46</td>
<td>44.66</td>
<td>44.9</td>
<td>43.99</td>
</tr>
<tr>
<td>700</td>
<td>67.1</td>
<td>64.81</td>
<td>62.35</td>
<td>61.52</td>
<td>61.48</td>
<td>60.64</td>
<td>60.72</td>
<td>59.73</td>
</tr>
<tr>
<td>900</td>
<td>86.9</td>
<td>84.37</td>
<td>83.05</td>
<td>81.17</td>
<td>82.13</td>
<td>80.2</td>
<td>81.11</td>
<td>79.3</td>
</tr>
</tbody>
</table>

It is seen from Table 6.2 and Figure 6.3 that the proposed ACO-DE with RS achieved lower makespan compared to ACO-RLS with RS scheduling and Proposed ACO with RS scheduling. ACO-DE scheduling with RS achieved 0.99% to 1.51% less makespan than ACO-RLS with RS. ACO-DE scheduling with RS achieved 2.09% to 2.95% less makespan than Proposed ACO with RS scheduling.
Table 6.3 Resource Utilization for 5 Resources

<table>
<thead>
<tr>
<th>Number of jobs</th>
<th>ACO with RS</th>
<th>Proposed ACO</th>
<th>Proposed ACO with RS</th>
<th>ACO-RLS with RS</th>
<th>ACO-DE with RS</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>83</td>
<td>86.01</td>
<td>84.27</td>
<td>88.13</td>
<td>84.68</td>
</tr>
<tr>
<td>300</td>
<td>82.9</td>
<td>86</td>
<td>84.61</td>
<td>88.09</td>
<td>84.99</td>
</tr>
<tr>
<td>500</td>
<td>86</td>
<td>88.59</td>
<td>87.28</td>
<td>90.33</td>
<td>87.37</td>
</tr>
<tr>
<td>700</td>
<td>83.5</td>
<td>86.92</td>
<td>85.24</td>
<td>88.81</td>
<td>85.31</td>
</tr>
<tr>
<td>900</td>
<td>81.9</td>
<td>84.56</td>
<td>83.91</td>
<td>86.11</td>
<td>84.23</td>
</tr>
</tbody>
</table>

It is seen from Table 6.3 and Figure 6.4 that the proposed ACO-DE with RS achieved better Resource Utilization compared to ACO-RLS with RS scheduling and proposed ACO with RS scheduling. ACO-DE scheduling with RS achieved 0.077% to 0.36% better Resource Utilization than ACO-RLS with RS. ACO-DE scheduling with RS achieved 0.14% to 0.67% better Resource Utilization than proposed ACO with RS scheduling.
Figure 6.4 Resource Utilization for 5 Resources

Table 6.4 Resource Utilization for 50 Resources

<table>
<thead>
<tr>
<th>Number of jobs</th>
<th>ACO</th>
<th>ACO with RS</th>
<th>Proposed ACO</th>
<th>Proposed ACO with RS</th>
<th>ACO-RLS with RS</th>
<th>ACO-DE with RS</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>83.6</td>
<td>87.49</td>
<td>85.51</td>
<td>89.17</td>
<td>85.85</td>
<td>89.39</td>
</tr>
<tr>
<td>300</td>
<td>83.1</td>
<td>86.3</td>
<td>84.81</td>
<td>87.87</td>
<td>85</td>
<td>88.19</td>
</tr>
<tr>
<td>500</td>
<td>81.1</td>
<td>83.4</td>
<td>82.89</td>
<td>85.42</td>
<td>83.29</td>
<td>85.62</td>
</tr>
<tr>
<td>700</td>
<td>82.2</td>
<td>86</td>
<td>83.43</td>
<td>87.51</td>
<td>83.62</td>
<td>87.61</td>
</tr>
<tr>
<td>900</td>
<td>82.7</td>
<td>85.76</td>
<td>84.11</td>
<td>87.35</td>
<td>84.39</td>
<td>87.44</td>
</tr>
</tbody>
</table>

It is seen from Table 6.4 and Figure 6.5 that the proposed ACO-DE with RS achieved better Resource Utilization compared to ACO-RLS with RS scheduling and Proposed ACO with RS scheduling. ACO-DE scheduling with RS achieved 0.24% to 0.45% better Resource Utilization than ACO-RLS with RS. ACO-DE scheduling with RS achieved 0.35% to 0.82% better Resource Utilization than Proposed ACO with RS scheduling.
6.4 CONCLUSION

Grid computing uses resource sharing and scheduling to ensure a high throughput by reducing resources waiting time. The new scheduling achieved grid environment’s high throughput computing. This is a NP-problem needs exponential time for a solution. So, a heuristic algorithm to locate a good solution in a reasonable time is developed.

In this new work DE and RLS algorithms was used to select optimal resources while running an application. ACO scheduled resources optimally. DE belongs to a class of GAs which use biology-inspired operations of crossover, mutation, and selection on a population to reduce an objective function over successive generations. Results showed a significant
improvement in applications performance when DE or RLS and ACO methods were combined for resource scheduling. The proposed ACO-DE with RS achieved lower Makespan compared to ACO-RLS with RS scheduling and Proposed ACO with RS scheduling. For 50 resources, ACO-DE scheduling with RS achieves 0.99% to 1.51% less makespan than ACO-RLS with RS. ACO-DE scheduling with RS achieves 2.09% to 2.95% less makespan than Proposed ACO with RS scheduling.