CHAPTER 3

ENHANCED ANT COLONY ALGORITHM HYBRID WITH PARTICLE SWARM OPTIMIZATION FOR GRID SCHEDULING

This chapter proposes new heuristic algorithms to solve grid scheduling problem. Two heuristic algorithms, based on Ant Colony Optimization and Particle Swarm Optimization are proposed. The optimization criteria, namely, flowtime and makespan are used to measure the quality of grid scheduling algorithm. Using the simulated benchmark instances, the results of different algorithms are analyzed and are compared with the optimization criteria.

3.1 ANT COLONY OPTIMIZATION

In recent years, biologically inspired metaphors have become an important and popular area of research within artificial intelligence. It particularly focuses on how the insects collectively solve the problem very easily, which is called swarm intelligence. The collective behavior of many relatively simple units is the fundamental field of artificial life. It creates an artificial system, which is controlled by the collective behavior. This work is inspired by biologists who have been studying the behavior of insects. They have uncovered the mechanisms controlling the foraging and food gathering behavior of ants. An ant’s path is directly influenced by the concentration of the pheromone trail values on the path. The pheromone is a specialized chemical substance which is laid in amounts and determined by the local
circumstances. These types of approaches are best suited for distributed systems, such as grid scheduling in grid computing environment.

3.1.1 Nature of Ants

The individual ant’s behaviors are very provincial. Each ant has a limited memory and the behavior has a large random component. When the ants act in a collective manner, they perform a variety of complex tasks with reliability and consistency. Regulating nest temperatures within limits of 1°C, searching particular area for food, forming bridges, working together to carry large items, building and protecting their nest and finding the shortest routes from the nest to a food source are some examples of ant’s collective behavior. These behaviors are achieved from the large number of individual ant’s interaction and their environment. The ants follow the stigmergy principle. It is an indirect coordination between the agents or actions by the same or different agent. It produces better results without any planning, control or communication between the agents. Ant’s action always depends on the response of the specific local environmental stimuli and not on the execution of some central plan. The environmental changes depend on two factors. First, the physical characteristics may be changed depending on the task related actions. The combined effect of these local task related changes can guide the growth of complex structure. Second, the ants deposit a value which causes environmental changes.

The depositions may cause the influences of subsequent behavior which is task related. This sign-based stigmergy has been developed by ants which use pheromones for a signaling system. Ant behavior, like sematectonic stigmergy (Theraulaz and Bonabeau 1995) and sign based stigmergy (Stickland et al 1992) are successfully simulated with computer models. The sign-based stigmergy is used in the grid scheduling model. It is based on how the ants find short routes from the source to food and also
consider the ways of selecting food sources depending on different values. The way by which ants organize their routes forms the base for QoS based ant algorithm for grid scheduling.

3.1.2 Basic Principles of Pheromone Update

Without using visual cues, the real ants are able to find the shortest paths between the sources and the nest with the help of pheromones. The ants are able to adopt the environmental changes easily using these pheromone values. A pheromone is an aromatic substance. Ants deposit a chemical trail on the ground as they walk and thus make the path by a trail of this substance. The following example shows how the ants reach the food source ‘F’ from the nest ‘N’ when more than one path is available from the nest to food source, for instance paths $a$ and $b$ in Figure 3.1(1). Assume that one path is shorter than all the other paths (Deneubourg and Goss 1989).

![Figure 3.1 Ants finding food using pheromone deposits](image-url)
Figure 3.1 shows the behavior of real ants. Currently no pheromone trails are available on the path. So, the first ant selects its path randomly as in Figure 3.1(1). A lot of ants move by a random choice of the available path with the probability of 0.5 each. When ants arrive at a bridge, they have to decide whether to turn left or right. Initially, no hint is available about the available path which is the shortest. So, the ants select their path randomly. Some of the ants decide to select left side and other ants select right side as shown in Figure 3.1(2). During their move, all ants deposit some amount of pheromone on the path. In general, the ants which use the shortest path reach their food more quickly than the other ants. Ants return on the shortest path to the nest more quickly than those which selected the longer path as shown in Figure 3.1(3).

The quantity of pheromone deposited on the path by the ants will be high in the shortest paths because a high number of ants will choose the shortest path on an average. After sometime, the amount of pheromone in the shortest path is higher than one of the longest paths. There after, when new ants arrive, they will prefer the shortest path because of the large amount of pheromone on the available paths. With the positive feedback effect, a large number of ants choose the shortest path. Some amount of pheromone on the path periodically evaporates. So, the longest path’s pheromones slowly evaporate and the trail on the longer path will weaken and eventually disappear. Finally, all ants tend to use the shortest path as shown in Figure 3.1(3).

Real ants are capable of adopting the environmental changes. After finding the shortest path, ants move on the shortest path, which connects a food source to the nest (Figure 3.1(3)). When an obstacle has appeared on the path, the ants which are in front of the obstacle are not able to follow the pheromone trail. Hence, they have to choose randomly and move both sides.
The ants which choose the shortest path around the obstacle will more rapidly reconstitute the interrupted pheromone trail, compared to those which choose the longer path. Thus, the shortest path will have the large amount of pheromone and a large number of ants will automatically select the shortest path for their travel.

### 3.2 ANT ALGORITHM

The grid scheduler’s aim is to allocate the tasks to the available nodes. To do this, the best match must be found from the list of available tasks to the list of available resources. The selection is based on prediction of the computing power of resources. There are a lot of problems need to be solved in this area. The grid scheduler must allocate the tasks to the resources efficiently. The efficiency depends on two criteria, namely, makespan and flowtime. These two criteria are very much important in the grid system. Makespan measures the throughput of the system while flowtime measures its QoS. Following assumptions are made before discussing the algorithm. Collection of independent tasks with no data dependencies is called as meta-task. The ant based algorithm is evaluated using the simulated execution time for a grid environment. Before starting grid scheduling, the expected execution time for each task on each machine must be estimated using the simulation technique and represented by an $ETC$ matrix.

#### 3.2.1 Simulation Model

The benchmark simulation model of grid computing environment is described by Braun et al (2001). The $ETC$ matrix calculation is based on task profiling and analytical benchmarking, discussed in Khokhar et al (1993), Maheswaran et al (1999) and Siegel et al (1997). In $ETC$ matrix, each row is used for representing jobs and each column relates to the available resources.
In general the ETC matrix will have $m \times n$ entries, where $m$ is the number of jobs and $n$ is number of available resources.

In grid environment, the job’s completion time depends on the execution time, the time taken to move the job to the particular resource, the time taken to move the necessary data to that resource and the time taken to move the final results to the corresponding user. This simulation model assumes that each value in the ETC matrix is considered with the above overheads. In the grid environment, all resources are not suited for all jobs. Only some resources are allowed for some special kind of jobs. So, some of the entries in ETC matrix may have infinity which means the corresponding jobs are not possible to run on those resources.

Task heterogeneity, resource heterogeneity and consistency are the metrics that are considered to generate different types of ETC matrix. The amount of variance among the possible execution time is defined as task heterogeneity. This simulation method defines two values, high and low for this variance. Resource heterogeneity is the variation of the execution time of a particular job across all the resources. Variance of the resource heterogeneity is also represented by high and low values. Consistent, inconsistent and partially consistent are the different type of consistencies that are used in this simulation model.

In the grid environment, sometimes, all the resources are of the same type but, only the processing speed of each resource is different; then this type of environment is called consistent. In a consistent ETC matrix if a resource $x$ executes a job $j$ faster than resource $y$, then all the jobs will be executed faster in resource $x$ than the resource $y$. This type of ETC matrix is called consistent. Inconsistent ETC matrix is generated when the environment is inconsistent. In the inconsistent environment, the resources are of different
In this environment, some of the resources performed very well in the jobs that have a huge amount of scientific computations but, they performed very slowly in data oriented jobs. But, some other resources performed well in the jobs that have large amount of data. Partially consistent matrix is also an inconsistent matrix but, it has a consistent sub matrix of a predefined size. This type of matrix simulates the computational grid environment that has some sub network and the same type of resources with different processing speeds.

The simulation model considered the above constraints of the three metrics, combined the three metrics and made it possible to generate twelve different possible ETC matrices randomly. The simulation model uses a base vector $B$ of $m \times 1$ size. Here, $m$ distinct random floating point numbers between 1 and $\phi_b$ are generated and stored in a vector $B$. Each value of $B(i)$ is multiplied by a uniform random number $x_{r,i,k}$ which has an upper bound $\theta_r$. $x_{r,i,k}$ is known as row multiplier. Each row of ETC matrix is defined by $ETC[i,j]$ equal to $B(i)$ multiplied by $x_{r,i,k}$, for $0 < k < n$. The above process is repeated to fill all the rows of the ETC matrix. Different base values are used to describe different tasks and resource heterogeneities. Task heterogeneity is defined as the value of $\phi_b$ in between 100 and 3000. When $\phi_b$ is 100, then the tasks are of low heterogeneity. When $\phi_b$ is 3000, then the tasks are of high heterogeneity. Similarly, the resource heterogeneity is described with the help of $\theta_r$. When $\theta_r$ is 10, then the resources are of low heterogeneity, if the value is 1000, then the machines are of high heterogeneity.

Consistent matrices are created; when each row of the consistent matrix is sorted independently that means the first resource has a higher speed than the last resource in that row. Inconsistent matrices have the values in random state. But, there is no need to sort the number row wise because each and every resource will execute faster for some jobs and slower for some
other jobs. After generating the $ETC$ matrices, the partially-consistent matrices are created by sorting each row's even column elements and by omitting the odd column elements. This means that the even column’s resources are consistent while the odd column’s resources are inconsistent.

### 3.2.2 Ant Colony Algorithm

The $ETC$ matrix has $M \times N$ entries, where $M$ is the number of independent tasks to be scheduled and $N$ is the number of resources which are currently available. Each task’s workload is measured by millions of instructions and the capacity of each resource is measured by Million Instructions Per Second (MIPS). The Ready time ($Ready_n$) indicates the time taken by resource $n$ to finish the previously assigned tasks. The completion time of $i^{th}$ task on $j^{th}$ machine is,

$$CT_{ij} = Ready_j + ETC_{ij}$$

$(3.1)$

$Max (CT_{ij})$ is the makespan of the complete schedule. Makespan is used to measure the throughput of the grid system. In general, the existing heuristic mapping can be divided into two categories, namely, immediate mode and batch mode. In immediate mode, the scheduler is always on ready mode. Whenever a new task arrives to the scheduler, it is immediately allocated to one of the existing resources, required by that task. However, each task is considered only once for matching and scheduling. In batch mode, the tasks and resources are collected and mapped at prescheduled time. In this mode, it takes a better decision because the scheduler knows the details of available tasks and resources. The proposed algorithms are heuristic based algorithms for batch mode.

The result of Fidanova and Durchova (2006) has four values (task, machine, starting time and expected completion time). Number of tasks
available for scheduling is always greater than the available number of machines in the grid. The resource $N_j$’s free time will be known using the function $\text{free}(j)$. The starting time of task $t_i$ on resource $N_j$ is,

$$\text{Ready}_i = \text{free}(j) + 1$$  \hspace{1cm} (3.2)

Then, the new value of $\text{free}(j)$ is the starting time plus $ETC_{ij}$. Minimization function $F_k$ is used to find out the best resource. Then, the following heuristic information $\eta_{ij}$ is used:

$$F_k = \max(\text{free}(j))$$ \hspace{1cm} (3.3)

$$\eta_{ij} = \frac{1}{\text{free}(j)}$$ \hspace{1cm} (3.4)

Using equation (3.4) the highest priority machine which is free earlier is found. Here, only one ant is used. The ant starts from random resource and task (they select $ETC_{ij}$ randomly $j^{th}$ resource and $i^{th}$ task). A list is maintained by the ant to store the task and resource which is to be selected next. At each iteration, ants calculate the minimized function $F_k$ ($k^{th}$ ant) and the pheromone level of the elements of the solutions is changed by applying the following update rule,

$$\tau_{ij} = \rho \tau_{ij} + \Delta \tau_{ij}$$ \hspace{1cm} (3.5)

where,

$$\Delta \tau_{ij} = \frac{1-\rho}{F_k}$$ \hspace{1cm} (3.6)
The rule $0 < \rho < 1$ models evaporation and $\Delta \tau_{ij}$ is an additional pheromone and it is different for different ACO algorithms. In this algorithm, two sets of tasks are maintained. One is a set of scheduled tasks and the other is a set of arrived and unscheduled tasks. The algorithm starts automatically, whenever the set of scheduled tasks become empty.

The first task to be performed and the machine in which it is performed are chosen by the equation (3.7):

$$P_{ij}^k = \frac{(\tau_{ij})^\alpha (\eta_{ij})^\beta}{\sum (\tau_{ij})^\alpha (\eta_{ij})^\beta}$$  (3.7)

where, $\eta_{ij}$ is the attractiveness of the move as computed by some heuristic information indicating a prior desirability of that move.

$\tau_{ij}$ is the pheromone trail level of the move, indicating how profitable it has been in the past to make that particular move (it represents, therefore, a posterior indication of the desirability of that move).

$P_{ij}^k$ is the probability to move from a state i to a state j, depending on the combination of the above two values.

$\alpha, \beta$ are the tuned parameters for the probability of the amount of the pheromone and the heuristic information.

Equation (3.7) has the disadvantage that all the columns in the probability matrix have the same probability value. This decides the best resource, but the task is chosen to be the first non zero value of the column. To overcome this disadvantage, the probability matrix $P_{ij}$ is modified. From all the possible scheduling lists, it is necessary to find the one having minimum makespan and use the corresponding scheduling list.
Here, two kinds of ETC matrices are formed, the first one consisting of currently scheduled tasks and the next consisting of tasks which have arrived but not scheduled. The scheduling algorithm is executed periodically. At the time of execution, it finds out the list of available resources (processors) in the grid environment, from the ETC matrix and starts scheduling. When all the scheduled tasks are dispatched to the corresponding resources, the scheduler starts scheduling over the unscheduled task. This guarantees that the machines will be fully loaded for maximum time.

Makespan is one of the optimization criterion of grid scheduling algorithm. Flowtime and resource utilization are the other optimization criteria (Xhafa et al 2007 and 2007a). These optimization criteria are used to measure the quality of the schedule.

### 3.2.3 Optimization Criteria

**Makespan** is the completion time of the latest task. It is defined as,

\[
\min_{\text{schedule}} \left[ \max \left\{ CT_{ij} \mid i \in \text{task and } j \in \text{resource} \right\} \right]
\]  

(3.8)

**Flowtime** is the sum of completion time of the tasks and is defined as,

\[
\min_{\text{schedule}} \left[ \sum_{j: \text{tasks}} CT_{ij} \mid i \in \text{task and } j \in \text{resource} \right]
\]  

(3.9)

Makespan and flowtime are very important parameters of the grid system. The grid system’s throughput is measured with the help of makespan parameter and the quality of grid system is measured with flowtime.
Fidanova and Durchova (2006), algorithm uses the heuristic information and pheromone trail update in probability calculation to find out the better resource and the better task for each iteration. The heuristic information considers the current load of each resource, and the pheromone trail calculation considers the heavily loaded resource’s current load in that iteration. Each time, the next resource and task are found with the help of the heuristic information, the pheromone trail is updated. But the heuristic information and the pheromone trail update considers only the resource related details. Hence by using this method, it is possible to select the better resource however the algorithm selects the task randomly. As the tasks are selected randomly, the local makespan of that particular resource may be increased. This results in the increase of overall makespan. When the random selection process in the Fidanova and Durchova (2006) algorithm is replaced with a technique, then the overall makespan can be minimized.

Improved Ant Colony algorithm (IAC) and Ant Colony Hybrid with PSO algorithm (ACHPSO) are the two heuristic algorithms proposed in the thesis. The proposed IAC algorithm uses a new pheromone updating rule and a new probability matrix calculation formula in order to overcome the disadvantages of the existing algorithms and thereby increase the efficiency of the existing ant colony algorithm. Further the optimization result is improved by combining the IAC algorithm with PSO algorithm.

3.3 PROPOSED IMPROVED ANT COLONY ALGORITHM (IAC)

Ant algorithm for scheduling initially started with the value of \( ETC_{ij} \) matrix. \( ETC_{ij} \) matrix defines the expected time taken by machine j to complete task i. Jobs considered here are independent of each other. \( ETC_{ij} \) matrix consists of \( N \times M \) entries where \( N \) is number of independent jobs and \( M \) is number of resources available in grid environment. The following steps clearly describe the proposed IAC algorithm.
Step 1: Collect all details about jobs and resources and calculate the expected completion time ($CT_{ij}$). $CT_{ij}$ is the completion time of $i^{th}$ job in the $j^{th}$ resource. It can be defined as:

\[
CT_{ij} = \text{Ready}_j + \text{ETC}_{ij}
\]  

(3.10)

where $\text{Ready}_j$ defines the ready time of the resource $j$. Maximum value of $CT_{ij}$ is considered as the makespan of the overall scheduling process. Free time of machine $j$ can be calculated using $\text{free}(j)$ function. $\text{free}(j)$ defines the free time of machine $j$. Beginning time of machine $j$ can be defined as:

\[
\text{Ready}_j = \text{free}(j) + 1
\]  

(3.11)

Completion time matrix is calculated using the $B_i$ value as given below:

\[
CT_{ij} = B_i + \text{ETC}_{ij}
\]  

(3.12)

where $B_i$ = Beginning time of $t_i$ on machine $m_j$.

Makespan of the grid can be calculated using the $CT_{ij}$ value. Makespan is used to measure the throughput of the grid system. Maximum $CT_{ij}$ value will be taken as the makespan of the grid system. A good algorithm should minimize the makespan value of the grid system.

Step 2: Initialization of value of parameters

- $\rho = 0.05$ (pheromone evaporation value)
- $\tau_0 = 0.01$ (initial pheromone deposit value)
- $\alpha = 1$ (importance of pheromone)
- $\beta = 2$ (importance of resource innate attribute)
- free $[0..m-1] = 0$ (one dimension matrix of size $m$)
- $k = m$ (number of ants = number of tasks)
Step 3: For each ant (to prepare the scheduling list) do the following step 4 and step 5

Step 4: Select the task (i) and resource (j) randomly.

Step 5: Repeat the following until all jobs are executed

a. Calculate the heuristic information \( r_{ij} \)

\[
\tau_{ij} = \frac{1}{\text{Free}(j)}
\]  \hspace{1cm} (3.13)

The \( r_{ij} \) formula helps to predict the more desirable machine in the available machine set. If a machine is free earlier, then the corresponding machine will be more desirable.

b. Calculate current pheromone trail value

\[
\Delta \tau_{ij} = \frac{(1-\rho)}{F_k}
\]  \hspace{1cm} (3.14)

where \( F_k = \text{max}(\text{free}(j)) \)

c. Update the pheromone trail matrix

The new pheromone updating rule is given by:

\[
\tau_{ij} = \left( \frac{\rho_i}{1+\rho_j} \right) \times \tau_{ij} + \left( \frac{\rho_i}{1+\rho_j} \right) \times \Delta \tau_{ij}
\]  \hspace{1cm} (3.15)

where \( \tau_{ij} \) - Pheromone trail level of the move

\( \rho_i \) - Pheromone evaporation value

\( \Delta \tau_{ij} \) - Additional pheromone
d. Calculate the probability matrix

\[
P_{ij} = \frac{\tau_{ij}^{\alpha} \cdot \eta_{ij}^{\beta} (1 / ETC_{ij})}{\sum \tau_{ij}^{\alpha} \cdot \eta_{ij}^{\beta} (1 / ETC_{ij})}
\]  

(3.16)

where

- \( ETC_{ij} \) - Expected Time to Complete Matrix;
- \( \tau_{ij} \) - Pheromone trail level of the particular path;
- \( \eta_{ij} \) - Heuristic information which replicates the attractiveness of the machine;
- \( \alpha \) - Importance of pheromone;
- \( \beta \) - Importance of resource innate attribute;

e. Select the task, with highest probabilities of \( i \) and \( j \) as the next task, to be executed on the machine \( j \).

**Step 6:** Find the best feasible solution using the scheduling list of all ants.

Increasing the pheromone levels associated with a chosen set of good solutions makes the algorithm faster to converge to a solution. Hence on modifying the pheromone updating rule and probability matrix, the solution is converging in a fast manner than the existing ACO algorithm. The probability matrix calculation uses the same transition rule to select the job which is to be executed next in the machine.

### 3.4 PROPOSED ANT COLONY HYBRID WITH PSO ALGORITHM (ACHPSO)

The proposed improved ant colony based algorithm (IAC) is hybrid with particle swarm optimization algorithm for overcoming the issues in the grid and to have better makespan. When compared with other algorithms, IAC algorithm reduces the makespan of the grid system and has achieved
high throughput computing. Convergence speed is an issue in the existing ant colony algorithms. Existing algorithms use multiple agents (ants) for searching the optimal solution in the search space. Number of agents should be less than the number of tasks. Existing Ant colony algorithms are hybrid only with simple local search algorithm (Kousalya and Balasubramanie 2009) such as Move Algorithm, Swap Algorithm, Move Minimum Completion Time Algorithm, Move Maximum Algorithm and Move Top Algorithm.

Drawbacks of Ant colony algorithm with local search are, it produces low efficiency in execution time and makespan attribute view, it results in high makespan which reduces the overall throughput of the grid system and stand alone local search algorithm requires processed input and not able to produce optimal solution always.

Hence a new algorithm is designed mainly to achieve high performance computing as well as high throughput computing. It considers the expected execution time of the task with respect to all machines in the grid environment. PSO algorithm can further optimize the solution obtained by ant colony optimization algorithm. Hashing the previous output using modulo operator helps to couple the ant colony algorithm with particle swarm optimization algorithm.

3.4.1 Hashing

Hashing is performed for the output of IAC algorithm which is then given as input to the PSO algorithm, hence, ACHPSO is developed.

Hash function is applied on the machines, initially; all the machine numbers which satisfy the hash function will be updated in the hash table. In the IAC output, machine number will be same for some tasks (Same machine may be allotted to different tasks at different time). Hence, IAC output cannot be used as it is by the PSO algorithm which necessitates the use of
intermediate hash conversion. Each machine number will be processed by hash function, if it matches head value in hash table, the corresponding subsequent item value in that head list will be replaced by the machine number. The process is repeated for all the machine numbers. In the sample output of IAC shown in Table 3.1 task 1 and task 4 has the same machine number. Hence, hashing process replaces the similar machine numbers with different machine numbers based on hashing function. This is done with the help of machine number hash table shown in Table 3.2.

Table 3.1 Sample IAC Algorithm Output

<table>
<thead>
<tr>
<th>Task</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Machine</td>
<td>3</td>
<td>10</td>
<td>5</td>
<td>3</td>
<td>0</td>
<td>7</td>
<td>9</td>
</tr>
</tbody>
</table>

Table 3.2 Machine Number Hash Table

Machine 3 is allocated to Task 1 and Task 4 as per the output of IAC algorithm. By using hash function, task 1 is scheduled to machine 3 and
task 4 is scheduled to machine 19. Hence the moderated table for PSO is represented by Table 3.3.

**Table 3.3 IAC algorithm output after Hashing**

<table>
<thead>
<tr>
<th>Task</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Machine</td>
<td>3</td>
<td>10</td>
<td>5</td>
<td>19</td>
<td>0</td>
<td>7</td>
<td>9</td>
</tr>
</tbody>
</table>

### 3.4.2 Algorithm of Proposed ACHPSO

In this section, a heuristic approach based on IAC and PSO algorithms is adapted for solving task scheduling problem in grid environment. Each particle is represented by a possible solution, and the position vector is transformed from continuous variable to discrete variable. This approach aims to generate an optimal schedule so as to get the minimum completion time while scheduling the tasks. Steps of ACHPSO are as follows:

**Step 1-6** As the proposed Improved Ant Colony algorithm

**Step 7** Read all the solutions available in Tabu list of an Ant Algorithm

**Step 8** Do hashing to avoid repetition in the machine row. Hashed value of the solution is the initial solution

**Step 9** Initialize parameters $c_1, c_2$ and number of iterations

**Step 10** Initialize each particle position and velocity vector using the following formula

$$x^i_{k+1} = x^i_k + v^i_{k+1} \quad (3.17)$$

$$v^i_{k+1} = \omega_k v^i_k + c_1 r_1 (p^i_k - x^i_k) + c_2 r_2 (p^g_k - x^i_k) \quad (3.18)$$
Step 11 Calculate fitness value and evaluate particle’s local best and global best

Step 12 Update each particle’s velocity and position

Step 13 Find permutations according to the updated each particle’s position

Step 14 Evaluate each particle and update the personal best and global best

Step 15 Apply local search

Step 16 Repeat the steps from 12 to 15 until completion of all iterations

Step 17 Print the global best solution.

3.5 ALGORITHM FOR OPTIMIZATION CRITERIA

Optimization criteria, makespan and flowtime are calculated by the following algorithms to study throughput and quality of the ant algorithms.

Variables

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Makespan</td>
<td>Makespan value of current schedule</td>
</tr>
<tr>
<td>Local_flowtime_j</td>
<td>Flowtime of each resource</td>
</tr>
<tr>
<td>Flowtime</td>
<td>Flowtime of the current schedule</td>
</tr>
<tr>
<td>NT</td>
<td>Number of tasks</td>
</tr>
<tr>
<td>NP</td>
<td>Number of resources</td>
</tr>
</tbody>
</table>

Input output list of IAC / ACHPSO algorithm

Output makespan and flowtime
Calculate Optimizing_Criteria

// To calculate Makespan
Initialize makespan = First Task’s completion time in the output list
For i = 2 to NT in the output list do
    If makespan < Task_i completion time in the output list do
        makespan = Task_i completion time in the output list
    End If
End For

// To calculate Flowtime
For i = 1 to NT in the output list do
    Local_flowtime[resource_i in the output list] =
    Local_flowtime[resource_i in the output list] +
    Task_i completion time in the output list
End For
Initialize flowtime = Local_flowtime[1]
For j = 2 to NP do
    If Local_flowtime[j] > flowtime
        flowtime = Local_flowtime[j]
    End if
End For

End Optimizing_Criteria

3.6 INITIALIZATION AND EXECUTION OF THE ALGORITHM

ACO algorithm has many parameters, which seem to interact in a fairly complex way, both with each other and with the specific problem class under investigation. Due to the time taken for a decent length run of the whole ACO algorithm, and also to the stochastic nature of the approach, finding the
optimal values for these parameters is a complex and time-consuming task, Ritchie and Levine (2003).

The parameter $\alpha$ determines the extent to which pheromone information is used as the ants build their solution. Pheromone is critical for the success of this algorithm, and having experimented with values from 1 to 10, it seems that this algorithm works best with the value of 1 for all the problems. $\beta$ determines the extent to which heuristic information is used by the ants. Values from 1 to 10 are tested, and the value 9 worked well for most problem types. But tests conducted on longer runs show that lower values produce better long term results, hence, the optimum value of $\beta$ is 2 for most cases.

In Fidanova and Durchova (2006), algorithm, the initial pheromone evaporation value $\rho$ is 0.05 and the tuned parameter $\alpha$ and $\beta$ values are 1. The initial value of pheromone deposit ($\tau_0$) is 0.01. Same values are used to initialize the parameters in the proposed algorithms except for the value of $\beta$ which is set as 2. Different values for parameters $\alpha$ and $\beta$ are investigated and the values 1 and 2 is found to be more optimal. Hence, these values are used to initialize the parameters in the proposed algorithm. The number of ants ($k$) used in the proposed algorithm is 2. The variable $free$ is a one dimensional matrix of size $n$ and the value is zero because, the proposed algorithm assumes that, all the resources are available only for grid scheduling.

After initialization, the proposed IAC algorithm runs as follows. The two ants run simultaneously. The first ant starts to find a solution. The algorithm first calculates the heuristic information $\eta_{ij}$. Then, it calculates the probability matrix by using the heuristic information $\eta_{ij}$, $ETC_{ij}$ and initial pheromone value $\tau_0$. After calculating the probability matrix, the algorithm finds out the maximum value $P_{ij}$ in the probability matrix $P$. 
After finding the initial task and its resource, the \((Task_i, resource_j, free(j), CT_{ij})\) values are added to the output list. Now \(Task_i\) is added to the scheduled list and is removed from the unscheduled list. The \(Task_i\)\(^{th}\) row is also removed from the probability matrix. This means, \(Task_i\) will not be considered for future scheduling. Now, the algorithm finds out the next task and its corresponding resources using the new probabilistic matrix. The free matrix, heuristic information, minimizing information, pheromone trail update and pheromone matrix are calculated and the calculated values are used to calculate the new probabilistic value of the unscheduled tasks. As per the new probabilistic value, the \(Task_i\) and \(resource_j\) are selected. On each iteration, one of the tasks is scheduled and that task is moved from unscheduled list to scheduled list. The above process repeats until the unscheduled list becomes empty. After the first ant finishes its tour, the next ant starts and finds out another set of solutions. When two ants finish their tour, the algorithm \(Optimization\_criteria\) calculates the makespan and flowtime of the two ant’s solution. The algorithm selects a better feasible solution from the two ant’s solution. The ant which yields the minimum makespan and minimum flowtime is considered as the better feasible solution of all the available solutions.

3.7 EXPERIMENTAL SETUP

The existing algorithms namely, OLB, MET, MCT, MaxMin, GA, Fidanova and Durchova (2006) and Kousalya and Balasubramanie (2008) are coded in JDK 1.6 and executed on a IBM compatible PC with Windows XP platform, Dual core - 2.33 GHz clock speed processor and 2 GB RAM. The proposed algorithms in this research work are also implemented in the same platform.

In grid computing, algorithms can be evaluated in two different ways. One, execution of the algorithm in a real grid environment and the
other is with a simulation model that simulates the grid environment. In the proposed research work simulation based approach is used. A simulated grid environment is created with the help of benchmark simulation model by Braun et al (2001). The results of the proposed and existing algorithms are compared for the same set of tasks and resources. This ensures identical experimental setup.

3.8 SIMULATION RESULTS AND DISCUSSION

Simulated benchmark instances in Braun et al (2001) are used to test the algorithms. Using this simulation model, 100 instances of ETC matrices are used, each with 12 possible types. The simulation model assumes that 16 resources are currently available in the grid environment and the scheduler wants to schedule 512 tasks. These $100 \times 12$ matrices are used in this thesis for testing the existing and proposed algorithms. Instances are labeled as $u_x_{yy}zz.k$ where,

- $u$ is a uniform distribution, used to generate the matrix.
- $x$ is a type of consistency which takes any one of the following:
  - $c$ consistent (if a resource $r_i$ executes a task $t_i$ faster than $r_j$).
  - $i$ inconsistent (a resource $r_i$ executes some tasks faster and some tasks slower than the other resources).
  - $p$ partially consistent (it contains consistent sub matrix).
- $yy$ is used to indicate the heterogeneity of the tasks ($hi$-high, $lo$-low).
- $zz$ is used to indicate the heterogeneity of the resources ($hi$-high, $lo$-low).
- $k$ is used to denote the version.
The proposed methods are deterministic. Their execution times are given by their time complexity. Hence, the hardware and the software configuration used are irrelevant. The results of the proposed IAC and ACHPSO algorithms are compared with the existing, immediate and batch mode scheduling algorithms. OLB, MET and MCT are the three existing immediate mode methods (Xhafa et al 2007a) that are implemented and compared with the proposed methods. Similarly, MaxMin (Xhafa et al 2007), ACFD (Fidanova and Durovova 2006) and ACKB (Kousalya and Balasubramanian 2008) are batch mode algorithms that are implemented and compared with the proposed methods.

Makespan and flowtime are computed using the above simulation model. The proposed methods run all the 100×12 instances and show the average values of 100 instances and they are given in Tables 3.4 and 3.5. Table 3.4 shows the average makespan values obtained from the immediate mode and batch mode algorithms. The graphical representation of the makespan values are shown in Figures 3.2 to 3.6. Figure 3.2 represents the graphical representation of average makespan values for all 12 types of instances. All the existing and proposed algorithms are normalized with OLB, by setting makespan of OLB value to one. Normalized makespan values are used for comparison, because the scales of makespan value have large variations for different types of instances. From Table 3.4 and Figure 3.2, it is evident that the proposed algorithms produce better results than the existing algorithms.

Table 3.4 shows that the proposed IAC algorithm yields better makespan results for 10 out of 12 considered instances. Makespan values of MET, ACFD and OLB algorithms are higher than other algorithms.
Table 3.4 Comparison of makespan values of various algorithms (in arbitrary time units)

<table>
<thead>
<tr>
<th>Instances</th>
<th>MakeSpan of Existing Algorithms</th>
<th>MakeSpan of Proposed Algorithms</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>MET</td>
<td>OLB</td>
</tr>
<tr>
<td>u_c_hibi.1</td>
<td>46264031</td>
<td>15084621</td>
</tr>
<tr>
<td>u_p_hibi.1</td>
<td>23421672</td>
<td>20998802</td>
</tr>
<tr>
<td>u_i_hibi.1</td>
<td>4599192</td>
<td>26954840</td>
</tr>
<tr>
<td>u_c_hilo.1</td>
<td>3044883</td>
<td>993800</td>
</tr>
<tr>
<td>u_p_hilo.1</td>
<td>2428189</td>
<td>2200214</td>
</tr>
<tr>
<td>u_i_hilo.1</td>
<td>716891</td>
<td>3214221</td>
</tr>
<tr>
<td>u_c_lohi.1</td>
<td>3976114</td>
<td>1296489</td>
</tr>
<tr>
<td>u_p_lohi.1</td>
<td>3196962</td>
<td>2871731</td>
</tr>
<tr>
<td>u_i_lohi.1</td>
<td>955417</td>
<td>3590920</td>
</tr>
<tr>
<td>u_c_lolo.1</td>
<td>111741</td>
<td>36438</td>
</tr>
<tr>
<td>u_p_lolo.1</td>
<td>90053</td>
<td>80590</td>
</tr>
<tr>
<td>u_i_lolo.1</td>
<td>26385</td>
<td>112051</td>
</tr>
</tbody>
</table>
Figure 3.2 Comparison of makespan with normalized results (with OLB makespan set to one) (in arbitrary time units)
The running time of ACKB, ACFD algorithms are little high compared to immediate methods and MaxMin algorithm. On comparing the results of ACHPSO algorithm, all the 12 instances yield least makespan values when compared to all other algorithms.

Figures 3.3 to 3.6 show the comparison of MET, OLB, MCT, MaxMin, ACFD, ACKB and proposed IAC and ACHPSO algorithms in high task high machine, high task low machine, low task high machine and low task low machine heterogeneities. From these figures, the performance of the existing and proposed algorithms can be easily analyzed with respect to the values of makespan.

![Figure 3.3 Makespan of various algorithms for high task high machine (in arbitrary time units)](image-url)
Figure 3.4  Makespan of various algorithms for high task low machine (in arbitrary time units)

From the analysis, MET and ACFD algorithms yield high makespan in 8 out of 12 considered instances. MET algorithm always allocates the task to the resource which has the minimum execution time. Hence, MET algorithm has highest makespan value for consistent tasks. This is shown in Figures 3.3 to 3.6. From these figures, it is seen that ACFD is not suited for inconsistent tasks. The results also show that the proposed IAC algorithm consistently discovers shorter makespan than the other exiting approaches for most cases of ETC matrix. This is achieved mainly due to modification done in the pheromone updating rule and inclusion of ETC in the calculation of probability matrix. IAC hybrid with PSO helps the system to converge faster thus results in reduced makespan. ACHPSO algorithm has produced reduced makespan in all the 12 instances considered.
Figure 3.5  Makespan of various algorithms for low task high machine (in arbitrary time units)

Figure 3.6  Makespan of various algorithms for low task low machine (in arbitrary time units)
Considering high task high machine heterogeneities, the makespan of the proposed ACHPSO has shown a reduction of 84.8% than ACFD and 83% than MET algorithms. The makespan value of the proposed ACHPSO is 51% and 43.6% less than ACKB and MCT algorithms respectively. Though the proposed IAC algorithm yields decreased makespan in 10 out of 12 instances, the makespan value is 37.6% more than the ACHPSO algorithm. Similar results are noted for other task and resource heterogeneities which are given in the Table 3.5.

**Table 3.5 Percentage Reduction in Average Makespan when Compared to ACHPSO**

<table>
<thead>
<tr>
<th>Heterogeneities</th>
<th>MET</th>
<th>OLB</th>
<th>MCT</th>
<th>MaxMin</th>
<th>ACFD</th>
<th>ACKB</th>
<th>IAC</th>
</tr>
</thead>
<tbody>
<tr>
<td>High task</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>High machine</td>
<td>83</td>
<td>79.9</td>
<td>43.6</td>
<td>57.2</td>
<td>84.8</td>
<td>51</td>
<td>37.7</td>
</tr>
<tr>
<td>Low machine</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>High task</td>
<td>74.8</td>
<td>75.7</td>
<td>27.4</td>
<td>48.5</td>
<td>77.6</td>
<td>24</td>
<td>10.9</td>
</tr>
<tr>
<td>Low machine</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Low task</td>
<td>73.7</td>
<td>72.4</td>
<td>24.1</td>
<td>46.3</td>
<td>75.8</td>
<td>20.3</td>
<td>12.9</td>
</tr>
<tr>
<td>High machine</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Low task</td>
<td>78.2</td>
<td>76.3</td>
<td>33.6</td>
<td>53.1</td>
<td>77.2</td>
<td>31.7</td>
<td>22.1</td>
</tr>
<tr>
<td>Low machine</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

From the Table 3.5, it is found that the average makespan of the proposed ACHPSO algorithm has shown a reduction of 77% from MET, 78% from ACFD, 76% from OLB, 51% from MaxMin, 32% from MCT, 31.7% from ACKB and 20.88% from the proposed IAC algorithm for all the 12 considered instances.
Makespan results of consistent, partially consistent and inconsistent instances are shown in Figures 3.7, 3.8 and 3.9 respectively. MET algorithm is not suitable for consistent and partially consistent instances. ACFD algorithm has produced poor results in most of the instances. OLB algorithm is not suited for partially consistent and inconsistent instances as the algorithm developed for consistent matrices. The proposed IAC and ACHPSO algorithms have produced least makespan in consistent, partially consistent and inconsistent instances.

![Figure 3.7 Makespan of various algorithms for Consistent high task high machine heterogeneities (in arbitrary time units)](image)

**Figure 3.7** Makespan of various algorithms for Consistent high task high machine heterogeneities (in arbitrary time units)

The other optimization criterion considered in the proposed research work is flowtime. The flowtime of the existing methods and the proposed methods are listed in Table 3.6. Using flowtime, the user knows how long the algorithm has utilized the grid environment for the given set of instances. Figures 3.10 to 3.13 show the comparison of flowtime of existing and proposed algorithms considered in the present research.
Figure 3.8 Makespan of various algorithms for Partially Consistent high task high machine heterogeneities (in arbitrary time units)

Figure 3.9 Makespan of various algorithms for Inconsistent high task high machine heterogeneities (in arbitrary time units)
Table 3.6 Comparison of flowtime values of various algorithms (in arbitrary time units)

<table>
<thead>
<tr>
<th>Instances</th>
<th>Flowtime of the Existing Algorithms</th>
<th>Flowtime of the Proposed Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>MET</td>
<td>OLB</td>
</tr>
<tr>
<td>u_c_hii.1</td>
<td>11602562678</td>
<td>3477309437</td>
</tr>
<tr>
<td>u_p_hii.1</td>
<td>3284952068</td>
<td>4901308260</td>
</tr>
<tr>
<td>u_i_hii.1</td>
<td>767868037</td>
<td>6182503044</td>
</tr>
<tr>
<td>u_c_hilo.1</td>
<td>118879901</td>
<td>35743222</td>
</tr>
<tr>
<td>u_p_hilo.1</td>
<td>34838454</td>
<td>51265883</td>
</tr>
<tr>
<td>u_i_hilo.1</td>
<td>8362476</td>
<td>64980267</td>
</tr>
<tr>
<td>u_c_lohi.1</td>
<td>38896314</td>
<td>116702514</td>
</tr>
<tr>
<td>u_p_lohi.1</td>
<td>111962928</td>
<td>165690163</td>
</tr>
<tr>
<td>u_i_lohi.1</td>
<td>36346727</td>
<td>208358045</td>
</tr>
<tr>
<td>u_c_lolo.1</td>
<td>3956937</td>
<td>1191636</td>
</tr>
<tr>
<td>u_p_lolo.1</td>
<td>1177792</td>
<td>1718254</td>
</tr>
<tr>
<td>u_i_lolo.1</td>
<td>281169</td>
<td>2164372</td>
</tr>
</tbody>
</table>
Figure 3.10  Comparison of flowtime of various algorithms for high task high machine (in arbitrary time units)

Figure 3.11  Comparison of flowtime of various algorithms for high task low machine (in arbitrary time units)
Figure 3.12  Comparison of flowtime for various algorithms for low task high machine (in arbitrary time units)

Figure 3.13  Comparison of flowtime for various algorithms for low task low machine (in arbitrary time units)
Figures 3.10 to 3.13 prove that MET, ACFD and OLB produce very large flowtime for all the instances. The MET algorithm does not consider the resource’s workload. All the tasks are allocated only to the high speed resource. Hence, all the four consistent matrices in Table 3.6 produce maximum flowtime for MET algorithm. The proposed IAC and ACHPSO algorithms have good flowtime values when compared to other algorithms. OLB algorithm yields poor results for 2 out of 12 instances and ACFD produces poor results for 6 out of 12 instances. From the result it shows that ACFD algorithm is not suited for inconsistent and partially consistent matrices.

3.9 CONCLUDING REMARKS

Average makespan of the proposed ACHPSO algorithm has shown a reduction of 77% from MET, 78% from ACFD, 76% from OLB, 51% from MaxMin, 32% from MCT, 31.7% from ACKB and 20.88% from the proposed IAC algorithm for all the 12 considered instances. From the results it is clear that the proposed ACHPSO algorithm produces better results by producing reduced makespan and flowtime when compared to other algorithms as well as the proposed IAC algorithm. The proposed IAC and ACHPSO algorithms are best suited for consistent, partially consistent and inconsistent instances.