An Intelligent Water Drop Algorithm for Optimizing Task Scheduling in Grid Environment

Sornapandy Selvarani[1] and Gangadharan Sadhasivam[2]
[1]Department of Information Technology, Tamilnadu College of Engineering, India
[2]Department of Computer Science and Engineering, PSG College of Technology, India

Abstract: *The goal of grid computing is to provide powerful computing for complex scientific problems by utilizing and sharing large scale resources available in the grid. Efficient scheduling algorithms are needed to allocate suitable resources for each submitted task. So scheduling is one of the most important issues for achieving high performance computing in grid. This paper addresses an approach for optimizing scheduling using a nature inspired Intelligent Water Drops (IWD) algorithm. In the proposed approach IWD algorithm is adopted to improve the performance of task scheduling in grid environment. The performance of Ant Colony Optimization (ACO) algorithm for task scheduling is compared with the proposed IWD approach and it is proved that task scheduling using IWD can efficiently and effectively allocate tasks to suitable resources in the grid.*

Keywords: *grid computing, IWD, task scheduling, ACO*.

Received January 24, 2013; accepted March 19, 2014; Published online December 23, 2015

1. Introduction

Computation grid technologies [1] are a new trend and appear as a next generation of the distributed heterogeneous system. It combines physical dynamic resources and various applications together. Task scheduling is a challenging problem in grid computing environment [2]. If large numbers of tasks are computed on the geographically distributed resources, an efficient scheduling algorithm should be adopted to allocate suitable resources for all the submitted tasks to compute the tasks within minimum time [6].

Scientist has realized that natural creatures and systems can be used as valuable sources of inspiration for developing new intelligent systems and algorithms [12]. Particle swarm optimization [3], Ant Colony Optimization (ACO) [4], Artificial neural networks [5] are some of the highly used technologies, for proposing new optimization algorithms, which partially or fully follow actions and reactions that happen in natural systems.

Among the most recent nature-inspired swarm-based optimization algorithms is the Intelligent Water Drops (IWD) algorithm.

It was first introduced by Kennedy and Eberhart [7], in which it was used to solve Travelling Salesman Problem (TSP). IWD algorithm also has been successfully applied to solve multiple knapsack problem [8] and to solve the n-queen problem [9].

In this paper we propose an algorithm based on *IWD* technique for allocating suitable resources to the submitted tasks in a grid environment. *IWD* algorithm is based on the dynamic of river systems, actions and reactions that happen among the water drops in rivers It can be used for either maximization or minimization problems. The solutions are incrementally constructed by *IWD* algorithm. So, it is a population-based constructive optimization algorithm.

IWD algorithm has been used for optimizing task scheduling in grid environment and it is called as IWDGS. Set of tasks are submitted to available resources in the grid. The problem is to allocate the submitted tasks to suitable resources that are available in the grid. So that, the execution time and cost of execution is minimized.

The next section discusses about how ACO algorithm is used for task scheduling in grid. Section 3 discusses about some natural actions that happen in rivers and gives how ideas of natural water drops are used to develop *IWD* algorithm and it also gives the mathematical modelling of *IWD*. Section 4 introduces *IWD* algorithm, which is used for solving our scheduling problem. How *IWD* algorithm is used for solving task scheduling in grid is discussed in section 5. Section 6 presents experimental results and discussions. Section 7 concludes the paper and suggests future works.

2. ACO Technique for Task Scheduling in Grid

Generally, when the jobs are submitted to the grid system at different times, they have different time steps. Therefore, they consume different resources. Therefore, processing is performed within different execution times. ACO algorithm is used to solve the above problem by considering the requirements of each job which is independent from other jobs and where only one of the job process in any unit times [11].

ACO is inspired by a colony of artificial ants cooperate in foraging behavior. This behavior enables ants to find the shortest paths between food sources and their nest. In fact, they deposit a chemical pheromone tail on the ground after they walk from the nest to food sources and vice versa. Hence, they choose the way with higher probability paths, which are marked by stronger pheromone concentrations. [10, 15]. Procedure for ACO algorithm as shown in Algorithm 1:

Algorithm 1: Procedure for ACO algorithm.

```
Procedure ACO
Begin
  Initialize parameters, the pheromone trails
  While (stopping criterion not satisfied) do
  Each ant position starts at starting machine
    While (stopping when every ant has build a
          Solution) do
        For each ant do
          Chose next machine by applying
          The state transition rate
        End for
    End while
    Update the pheromone
  End while
End
```

3. Basics of IWD

3.1. Concept of Natural Water Flow in Rivers

In natural rivers we can see flowing of water from one place to another through a path. The paths of water flow have been created by a swarm of water drops. It's also observed that the constructed path seems to be an optimal path in terms of distance travelled by the water [7].

One of the important properties of flowing water is its velocity. Water drops flowing from one place to another will carry some amount of soil along with it. The soil is usually transferred from fast parts of the path to slow parts of the river. The carried soils will be unloaded in the slower beds of the river.

The following properties are assumed for a flowing water drop in a river:

- A high velocity water drop removes more soil than a low velocity water drop.
- The velocity of a water drop increases more on a path with low soil than a path with high soil.
- Water chooses an easier path with less soil than a harder path with more soil.

3.2. Mathematical Modelling of the Actions and Reactions of IWD

Based on the properties discussed above Niu *et al*. [13] suggested an intelligent water drop which possesses a few properties of a natural water drop. Two important properties of *IWD* are:

- The soil a water drop carries: $Soil_{IWD}$.
- The velocity of the water drop: $Velocity_{IWD}$.

The above properties may change during the flow of water. From technical point of view, an environment represents a problem that is to be solved.

The problem here is to find an optimal path for the flow of water in the river. *IWD* is assumed to flow from a source to a desired destination. From a given source to a desired destination there will be multiple paths. In case the desired destination is known, the solution to the problem is to find the shortest path from the source to the destination. If the destination is unknown the shortest path is obtained based on a desired measure like cost, time etc.

An *IWD* moves in discrete finite- length steps in its environment. An *IWD* moves from current location *i* to a new location *j* and the velocity $Velocity_{IWD}$ is increased by an amount of $\Delta Velocity_{IWD}$ which is nonlinearly proportional to the inverse of the soil between *i* and *j*.

$$\Delta Velocity_{IWD} \propto^{NL} \frac{1}{soil(i,j)} \tag{1}$$

One possible formula is given in Equation 2 in which the velocity of the *IWD* denoted by $\Delta Velocity_{IWD}(t)$ is updated by the amount of soil $soil(i, j)$ between the two locations *i* and *j*.

$$\Delta Velocity_{IWD}(t) = \frac{a_v}{b_v + c_v . soil^{2}(i,j)} \tag{2}$$

Here, the a_v, b_v and c_v and α^{NL} are user-selected positive parameters. Moreover, the *IWD*'s soil, *soil*(*IWD*), is increased by removing some soil of the path joining the two locations *i* an *j*. The amount of soil added to the *IWD*, $\Delta soil(IWD) = \Delta soil(i, j)$ is inversely (and nonlinearly) proportional to the time needed for the *IWD* to pass from its current location to the next location denoted by *time*(*i*, *j*, *IWD*).

$$\Delta soil(IWD) = \Delta soil(i,j) \propto^{NL} \frac{1}{time(i, j : IWD)} \tag{3}$$

One suggestion for the above formula is given in Equation 4 in which $time(i, j, vel^{IWD})$ is the time taken for the *IWD* with velocity vel^{IWD} to move from location *i* to *j*. The soil added to the *IWD* is calculated by:

$$\Delta soil(i,j) = \frac{a_s}{b_s + c_s . time^{2}(i, j : vel^{IWD})} \tag{4}$$

Where the parameters a_s, b_s and c_s and θ are user-selected positive numbers. The duration of time for the *IWD* is calculated by the simple laws of physics for linear motion. Thus, the time taken for the *IWD* to move from location *i* to *j* is proportional to the velocity of the *IWD*, velocity (*IWD*) and inversely proportional to the distance between the two locations, $d(i, j)$. More specifically:

$$time(i,j;IWD) \propto^{L} \frac{1}{velocity(IWD)} \quad (5)$$

Where α^{L} denotes linear proportionality. One such formula is given below, which calculates the time taken for *IWD* to travel from location *i* to *j* with velocity vel_{IWD}:

$$time(i,j;IWD) = \frac{HUD(i,j)}{vel^{IWD}} \quad (6)$$

Where a local heuristic function *HUD*(*i*, *j*) has been defined for a given problem to measure the undesirability of an *IWD* to move from one location to the next. Some soil is removed from the visited path between locations *i* and *j*. The updated soil of the path denoted by *soil*(*i*, *j*) is proportional to the amount of soil removed by *IWD* flowing on the path joining *i* to *j*, $\Delta soil(i,j) = \Delta soil(IWD)$, specifically:

$$soil(i,j) \propto^{L} \Delta soil(i,j) \quad (7)$$

One such formula has been used for *IWD* algorithm such that *soil*(*i*, *j*) is updated by the amount of soil removed by *IWD* from the path *i* to *j*.

$$soil(i,j) = \rho_o . soil(i,j) - \rho_n . \Delta soil(i,j) \quad (8)$$

Where ρ_o and ρ_n are often positive numbers between 0 and 1. The soil of *IWD* denoted by $soil^{IWD}$ is added by the amount *soil*(*i*, *j*) as shown below:

$$soil^{IWD} = soil^{IWD} + \Delta soil(i,j) \quad (9)$$

Another mechanism that exists in the behaviour of an *IWD* is that it prefers the paths with low soils on its beds to the paths with higher soils on its beds. To implement this behaviour of path choosing, a uniform random distribution is used among the soils of the available paths such that the probability of the *IWD* to move from location *i* to *j* denoted by *p*(*i*, *j*, *IWD*) is inversely proportional to the amount of soils on the available paths.

$$p(i,j;IWD) \propto^{L} soil(i,j) \quad (10)$$

The lower the soil of the path between locations *i* and *j*, the more chance this path has for being selected by *IWD* located on *i*. One such formula based on Equation 10 has been used in which the probability of choosing location *j* is given by:

$$p(i,j;IWD) = \frac{f(soil(i,j))}{\sum_{k} f(soil(i,j))} \quad (11)$$

Where $f(soil(i,j)) = \frac{1}{s + g(soil(i,j))}$.

The constant parameter ε_s is a small positive number to prevent a possible division by zero in the function *f* (.). The set *vc*(*IWD*) denotes the nodes that *IWD* should not visit to keep satisfied the constraints of the problem. The function *g*(*soil*(*i*, *j*)) is used to shift the *soil*(*i*, *j*) of the path joining nodes *i* and *j* toward positive values and is computed by:

$$g(soil(i,j)) = \begin{cases} if\ min_{l \in vc(IWD)}(soil(i,j) \geq 0) & soil(i,j) \\ else & g(soil(i,j)) = soil(i,j) - min_{l \in vc(IWD)}(soil(i,j)) \end{cases} \quad (12)$$

Where the function *min*() returns the minimum value of its arguments. The *IWDs* work together to find the optimal solution of a given problem. The problem is encoded in the environment of the *IWDs* and the solution is represented by the path that the *IWDs* have converged to. In the next section, the *IWD* algorithm is explained.

4. IWD Algorithm for Task Scheduling in Grid

4.1. IWD Algorithm

The *IWD* algorithm is represented in the form of a graph with *N* number of nodes and *E* number of edges. Each *IWD* begins by constructing its solution gradually by travelling on the nodes of the graph along the edges of the graph until *IWD* finally completes its solution, which is denoted by T_{IWD}. Each solution T_{IWD} is the set of edges that *IWD* has visited.

One iteration of *IWD* algorithm is finished when all *IWDs* complete their solutions. After each iteration, the iteration best solution is found based on a quality function among all solutions obtained by *IWDs* in the current iteration and it is denoted by T_{IB}. The T_{TB} is the total-best solution which is the best solution since the beginning of *IWD* algorithm, which has been found in all iterations.

The quality of solution of *IWD* can be denoted by $QUL(T_{IWD})$. Therefore, the iteration-best solution T_{IB} is given by:

$$T_{IB} = arg(max_{forall IWDs} QUL(T_{IWD})) \quad (13)$$

Where *arg*() returns its argument. The total-best solution T_{TB} is updated by the current-best solution T_{IB} as follows:

$$T_{TB} = \begin{cases} if\ QUL(T_{TB}) \geq QUL(T_{IB}) & T_{TB} \\ Otherwise & T_{IB} \end{cases} \quad (14)$$

At the end of each iteration of *IWD* algorithm, the amount of soil on the edges of the iteration-best solution T_{IB} is reduced based on the quality of the solution. Shah-Hosseini [13] suggested a method to update the *soil*(*i*, *j*) of each edge of the iteration-best solution T_{IB}.

$$soil(i,j) = \rho_s . soil(i,j) \quad (15)$$

$$-\rho_{IWD} \cdot \frac{1}{(N_{IB} - 1)} . soil_{IB}^{IWD} \forall (i,j) \in T_{IB} \quad (16)$$

Where $soil_{IB}^{IWD}$ represents the soil of the iteration-best *IWD*. The iteration-best *IWD* is the *IWD* that has constructed the iteration-best solution T_{IB} at the current

iteration. Number of nodes in the solution T_{IB} is denoted as $N_{IB.}$ ρ_{IWD} ranges from 0 to 1 which is the global soil updating parameter and ρ_s=(1+ρ_{IWD}) The whole process is repeated with new *IWDs* but with the same soils on the paths of the graph. The algorithm is terminates when it completes all its iterations or when it achieves the expected quality measure.

IWD algorithm involves a number of parameters both static and dynamic.

The parameters which remain constant throughout the execution of the algorithm are assumed to be static and those parameters whose values changes dynamically during its execution are said to be dynamic parameters.

The values of the static parameters used in this paper are the same used by Niu *et al*. [14]. *IWD* algorithm as shown in Algorithm 2:

Algorithm 2: *IWD* algorithm.

1. *All the static and dynamic parameters used in the algorithm are initialized as follows*:

   a. *Initialization of Static Parameters:*

   - *The graph* (*N*, *E*) *of the problem is given to the algorithm, which contains Nc nodes.*
   - *The quality of the total-best solution T_{TB} is initially set to the worst value: QUL(T_{TB})= −∞.*
   - *The maximum number of iterations $iter_{max}$ is specified by the user and the algorithm stops when it reaches $iter_{max}$.*
   - *The iteration count $iter_{count}$, which counts the number of iterations, is set to zero.*
   - *The number of water drops N_{IWD} is set to a positive integer value. This number should at least be equal to two. However, it is usually set to the number of nodes Nc of the graph.*
   - *Velocity updating parameters a_v and c_v==1 and b_v=0.01.*
   - *Soil updating parameters a_s and c_s==1 and b_s= 0.01.*
   - *The local soil updating parameter is ρ_n= 0.9.*
   - *The global soil updating parameter is ρ_{IWD}= 0.9.*
   - *The initial soil on each edge of the graph is denoted by the constant InitSoil such that the soil of the edge between every two nodes i and j is set by soil*(*i, j*)=*InitSoil. Here, InitSoil*=10000.
   - *The initial velocity of each IWD is set to InitVel. Here, InitVel=200*

   b. *Initialization of Dynamic Parameters:*

   - *Every IWD has a visited node list Vc*(*IWD*), *which is initially empty*: *Vc*(*IWD*)={ }.
   - *Each IWD's velocity is set to InitVel.*
   - *All IWDs are set to have zero amount of soil.*

2. *Spread IWDs randomly on the nodes of the graph as their first visited nodes.*
3. *Update the visited node list of each IWD to include the nodes just visited.*
4. *Repeat steps* 5.1 *to* 5.4 *for those IWDs with partial solutions.*
5. 5.1. *For IWD residing in node i, choose the next node j, which doesn't violate any constraints of the problem and is not in the visited node list V_c*(*IWD*) *of IWD, using the following probability $p_i^{IWD}(j)$.*

$$p_i^{IWD}(j)=\frac{f(soil(i,j))}{\sum_{k\in vc(IWD)} f(soil(i,j))} \quad (17)$$

Such that:

$$f(soil(i,j))=\frac{1}{\varepsilon_s+g(soil(i,j))}$$

And

$$g(soil(i,j))=\begin{cases} If\ min_{l\in vc(IWD)}(soil(i,j)\geq 0)\ soil(i,j) \\ else \qquad soil(i,j)-min_{l\in vc(IWD)}(soil(i,j)) \end{cases}$$

Then, add the newly visited node j to the list V_c(*IWD*).

5.2. *For each IWD moving from node i to node j, update its velocity vel_{IWD}*(*t*) *by:*

$$vel_{IWD}(t+1)=vel_{IWD}(t)+\frac{a_v}{b_v+c_v.soil^2(i,j)} \quad (18)$$

Where vel_{IWD}(*t*+1) *is the updated velocity of IWD.*

5.3. *For IWD moving on the path from node i to j, compute the soil Δsoil*(*i, j*) *that IWD loads from the path by:*

$$\Delta soil(i,j)=\frac{a_s}{b_s+c_s.time^2(i,j;vel^{IWD}(t+1))} \quad (19)$$

Such that:

$$time(i,j;vel^{IWD}(t+1))=\frac{HUD(j)}{vel^{IWD}(t+1)}$$

Where the heuristic undesirability HUD(*j*) *is defined appropriately for the given problem.*

5.4. *Update the soil soil*(*i, j*) *of the path from node i to j traversed by that IWD, and also update the soil that IWD carries soil ^{IWD} by:*

$$soil(i,j)=(1-\rho_n).soil(i,j)-\rho_n\Delta soil(i,j)$$
$$soil^{IWD}=soil^{IWD}+\Delta soil(i,j) \quad (20)$$

6. *Find the iteration-best solution T_{IB} from all the solutions T_{IWD} found by IWDs using:*

$$T_{IB}=arg\ max_{\forall T^{IWD}} QUL(T_{IWD}) \quad (21)$$

Where function QUL(.) *gives the quality of the solution.*

7. *Update the soils on the paths that form the current iteration-best solution T ^{IB} by:*

$$soil(i,j)=(1+\varepsilon_{IWD}).soil(i,j)-\varepsilon_{IWD}.\frac{1}{(N_{IB}-1)}.soil_{IB}^{IWD}\forall(i,j)\in T_{IB} \quad (22)$$

Where N_{IB} is the number of nodes in solution T_{IB}.

8. *Update the total best solution T_{TB} by the current iteration-best solution T_{IB} using:*

$$T_{TB}=\begin{cases} If QUL(T_{TB})\geq QUL(T_{IB}) & T_{TB} \\ Otherwise & T_{IB} \end{cases} \quad (23)$$

9. *Increment the iteration number by $Iter_{count}$= $Iter_{count}$*+1. *Then, go to step 2 if $Iter_{count}$< $Iter_{max}$.*
10. *The algorithm stops here with the total-best solution T_{TB}.*

5. Task Scheduling in Grid using IWD Algorithm

The aim of the proposed algorithm is to use a new technique to select optimal resources from the grid which can find out the optimal resources to process the tasks and to improve the overall performance of the system in terms of execution cost and tardiness time.

The approach is to develop a scheduling algorithm within the objective to minimize the total execution cost and total tardiness time of the tasks based on *IWD* algorithm.

The problem is stated as follows: A set of n tasks are available for processing on available set of m resources. Each j^{th} task has a processing time P_j, an arrival time a_j, a release time r_j and expected time E_j. All the above values are assumed to be positive integers.

Let C_{ij} be the completion time of the operation of job j on machine i. It is denoted as:

$$C_{ij} = a_j + r_j + P_{ij} \tag{24}$$

The tardiness of the j^{th} task in machine i is given as:

$$T_{i,j} = max\,((C_{ij} - E_{ij}),0) \tag{25}$$

The objective is to minimize the maximal total tardiness time:

$$T_{i,j} = \Sigma_{i=1}^{m}(\Sigma_{j=1}^{n} T_{i,j}) \tag{26}$$

The grid scheduler architecture is shown below in Figure 1.

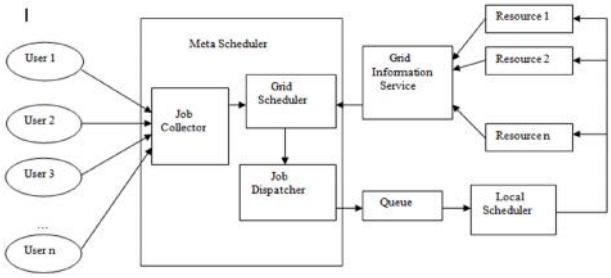

Figure 1. Architecture of grid scheduler.

5.1. The Proposed Scheduling Algorithm IWDGS

Generally, when the tasks are submitted to the grid system at different times, they have different time steps, therefore they use different resources. Therefore, the processing is performed within different execution times. The tasks are submitted dynamically to the grid system.

This paper proposes a scheduling algorithm based on *IWD* algorithm. *IWD* algorithm can be applied to solve both static and dynamic combination optimization problems.

IWD algorithm has the ability to search and modify the environment of the given problem. By doing this it constructs a solution incrementally. The problem is presented in the form of a graph and *IWD's* travel one node to another node through the edges of the graph.

A swarm of *IWD's* flows in the graph and they find the optimal solution using the local heuristic. In the proposed method *IWD* algorithm is applied to solve the scheduling problem in grid environment, Let M be a set of machines $\{m_1, m_2, m_3, ..., m_m\}$ and let J be a set of jobs $\{j_1, j_2, j_3, ..., j_n\}$ and $n>m$. Therefore, the graph $G=\{M, \{CT_{ij}\}_{mxn}\}$. The objective is to find the optimal resources for the jobs. It can minimize the total execution time and cost.

Each job has a sequence of operations $OP_k=\{OP_k|$ $k= 1, 2, ..., n\}$ and these 'n' jobs (all operations) have to be processed on 'm' machines. Job splitting is not allowed and the operations are non-premptable. Each machine only performs one operation at a time and each operation is performed only once on one machine. The algorithm is to find a feasible assignment of all operations on the given machines with optimized machine.

Our objective is to minimize the maximal total tardiness time of the schedules and to minimize the processing cost of submitted jobs. Maximal total tardiness time is calculated as defined in Equation 26 in section 4.

A graph with N nodes and E edges can be used to represent job scheduling in grid environment which resembles rivers as in *IWD* algorithm. The IWDGS algorithm is given below. Algorithm 3 has *NIWD_iter* iterations. In each iteration, *IWD's* travel from source node to the sink node in the graph. The path of an *IWD* can produce a feasible solution. The soils on the edges where *IWDs* pass, soils of *IWDs* and the velocities of *IWDs* are updated during the travelling of *IWDs*.

After each iteration, the soils on *NeliteIWDs* paths are updated. Next, a group of best solutions *SBD* are chosen and a combined local search is performed to further improve these solutions. After local search, a best iteration solution *SIB* is identified and the global best solution S^{TB} is updated. After all the iterations, another local search is performed on S^{TB}.

Algorithm 3: IWDGS.

```
Initialize an IWDs group A ; // A population of IWDs.
initialization ();
while (k < NIWD_iter) do
  for (each time step t) do
     for (each IWDg Č  A which feasible solution
    has not been discovered) do
      (i, j) = selectNextEdge(IWDg);
      Vel IWDg = updateVelocity(VelIWDg );
      Δsoil (i, j) = computeDeltaSoil((i ,j), IWD);
     soil (i, j)= updateEdgeSoil((i, j),Δsoil (i, j));
   soil IWDg = updateIWDSoil(soil IWDg,Δsoil( i, j);
   end for
   end for
   for (Nelite IWDs) do
        globalSoilPropagation();
  end for
```

SBD= setupBestSolutionGroup ();
SIB= combinedLocalSearch(SBD);
updatePathSoil(SIB);
update(STB);
k++;
end while
S^{TB} = *combinedLocalSearch*(S^{TB}).

initialization( ) //*This function initializes the static and dynamic parameters*

- *SelectNextEdge*(*IWDg*): For the g^{th} *IWD*, choose the next node to visit in the schedule operation list according to the probability calculation using Equation 17.
- *UpdateVelocity*(*IWDg Vel*): For the g^{th} *IWD* moving from node *i* to node *j* on the graph, update its velocity using Equation 18.
- *ComputeDeltaSoil*((*i*, *j*), *IWD*): For the g^{th} *IWD*, moving on the path from node *i* to *j*, calculate the amount of soil that it loads from the path using Equation 19.
- *updateEdgeSoil*((*i*, *j*), Δ*soil*(*i*, *j*)) and *updateIWDSoil*(*soil* ^{IWDg} Δ*soil*(*i*, *j*)) For the g^{th} *IWD*, update the soil of the edge it traversed and the soil contained in the *IWD* using Equation 20.
- *GlobalSoilPropagation*(): Update the soil of the edges included in the current elite *IWD's* solutions (*Nelite IWDs*).
- *SetupBestSolutionGroup*(): This is used for local search; a solution group *SBD* is set up for recording the best *NBD* solutions during the local search process.
- *CombinedLocalSearch*(*SBD*): This is a local search method which combines breadth search and depth search schemes in the search neighbourhood. The input of this function can be a group of solutions or a single solution and the output is an improved solution.
- *UpdatePathSoil*(S^{IB}): To update the soil of the path associated with the best iteration solution S^{IB} using Equation 22.
- *Update*(S^{TB}): Update the global best solution S^{TB} by the best iteration solution S^{IB} using Equation 23.

6. Experimental Results

The experimental simulation of IWDGS algorithm is implemented on an Intel Pentium 4 machine using the simulation Toolkit GridSim 4.1 which is Java based simulation toolkit. GridSim toolkit is designed basically for simulating resource management and task scheduling problems in grid environment. The simulation is designed based on grid simulation architecture described in section 4.

Below Tables 1 and 2 gives various attributes of the resources used and jobs submitted to the available resources.

Table 1. Resource attributes.

Attributes	Values
Number of Resources	5-10
Processor Speed (MIPS)	25-250
Bandwidth (Mbit)	10
Number of Processing Element in each Resource	1-5

Table 2. Job attributes.

Attributes	Values
Number of jobs Submitted	50-300
Length of Jobs (MI)	3000-9500
Number of Processing Element in each Resource	1-5

The values of velocity updating parameters are a_v and c_v =1.0 b_v=0.01 and soil updating parameters are a_s and c_s=1.0 and b_s=0.01. The values of local soil updating parameter is ρ_n=0.9 and global soil updating parameter is ρ_{IWD} =0.9

We have tested the proposed *IWD* algorithm for solving job scheduling in grid environment. Jobs are submitted to the available resources in the grid.

We have the experimented the proposed algorithm by submitting different numbers of jobs to different number of resources available in the grid. The results of the above experiment recorded the tardiness time and processing cost taken for executing different numbers of jobs submitted to different number of available resources in the grid.

We have compared the tardiness time and processing cost of IWDGS algorithm with ACO algorithm for scheduling jobs in grid environment. Tables 3 and 4 below show the results of the above comparisons. From the Tables it can be seen that the results of IWDGS algorithm is best when compared with the results of ACO algorithm. The comparison is proved with line graphs shown in Figures 2 and 3.

Table 3. Results of tardiness time of IWDGS and ACO algorithms for task scheduling in grid.

Number of Jobs (gridlets)	Tardiness Time (in Sec's)	
	IWDGS	**ACO**
25	165.19	226.36
50	320	429.8
75	484.87	659.34
100	658	912.53
125	818.04	1140.35
150	982.75	1376.38
175	1137.84	1590.51
200	1271.34	1734.52
225	1453	2041.1
250	1613.87	2270.25
275	1795	2522.02
300	1944.24	2730.12

Table 4. Results of processing cost of IWDGS and ACO algorithms for task scheduling in grid.

Number of Jobs (gridlets)	Processing Cost (in Rs.)	
	IWDGS	**ACO**
25	347.98	538.09
50	661	1021.86
75	1024.6	1559.85
100	1394	2158.01
125	1732.6	2671.06
150	2081.88	3229.15
175	2407	3721.53
200	2678.3	4153.55
225	3070	4773.30
250	3409.67	5310.76
275	3800	5616.06
300	4110.62	6390.37

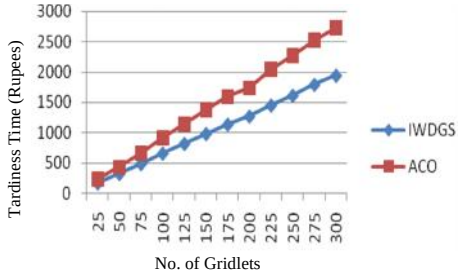

Figure 2. Tardiness time comparison of IWDGS and ACO algorithms.

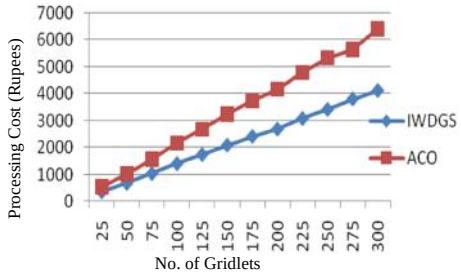

Figure 3. Processing cost comparison of IWDGS and ACO algorithms.

We have also compared utilization of resources in the grid for both ACO algorithm and IWDGS algorithm. The comparison graph for the above comparison is shown in Figure 4. The mean utilization for 10 resources is 70%.

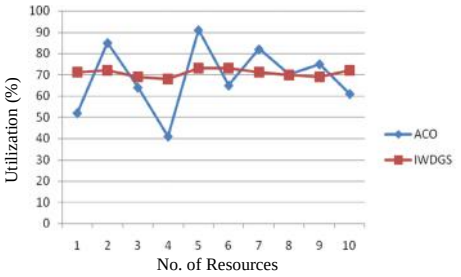

Figure 4. Resource utilization comparison of IWDGS and ACO algorithms.

The standard deviation for IWDGS algorithm is 1.75119 while the standard deviation for ACO algorithm is 15.3565.

So, we can come to a conclusion that IWDGS algorithm is keeping track the states of all the resources at each point in time so that more optimal decision can be made at each time.

7. Conclusions

This paper investigates an optimization algorithm named IWDGS. The proposed algorithm is presented and it is applied to solve job scheduling in grid environment. The algorithm is developed based on intelligent water drop optimization algorithm to allocate the user submitted jobs to the various resources available in the grid. The algorithm can select optimal resources to allocate submitted jobs such that it minimizes the maximal total tardiness time and processing cost.

The proposed algorithm was compared with ACO algorithm for scheduling jobs in grid. From the experimental results obtained we have proved that our proposed algorithm is much efficient than ACO algorithm for scheduling jobs in grid.

References

[1] Abraham A., Buyya R., and Nath B., "Nature's Heuristics for Scheduling Jobs on Computational Grids," available at: www.buyya.com/papers/nhsjcg.pdf, last visited 2000,.

[2] Cormen T., Leiserson, C., Rivest R., and Stein C., *Introduction to Algorithms*, MIT Press, Cambridge, 2003.

[3] Dorigo M. and Stutzle T., *Ant Colony Optimization*, Prentice-Hall, 2004.

[4] Foster I. and Kesselman C., *The Grid*: *Blueprint for a Future Computing Infrastructure*, Morgan Kaufman Publishers, 1999.

[5] Haykin S., *Neural Networks*, Prentice-Hall, 1999.

[6] Ijaz S., Munir E., Anwar W., and Nasir W., "Efficient Scheduling Strategy for Task Groups in Heterogeneous Computing Environment," *The International Arab Journal of Information Technology*, vol. 10, no. 5, pp. 486-492, 2013.

[7] Kennedy J. and Eberhart R., *Swarm Interllignece*, Morgan Kaufmann, 2001.

[8] Kirkpatrick S., Gelatt C., and Vecchi M., "Optimization by Simulated Annealing," *Science*, vol. 220, no. 4598, pp. 671-680, 2008.

[9] Koza, J., *Genetic Programming: On the Programming of Computers by means of Natural Evolution*, MIT Press, Massachusetts, 1992.

[10] Ku-Mahamud K. and Nasir H., "Ant Colony Algorithm for Job Scheduling in Grid Computing," *in Proceedings of the 4th Asian International Conference on Mathamatical/ Analytical Modelling and Computer Simulation*, Kota Kinabalu, pp. 40-45, 2010.

[11] Mandloi S. and Gupta H., "Adaptive Job Scheduling for Computational Grid based on ACO with Genetic Parameter Selection," *International Journal of Advanced Computer Research*, vol. 3, no. 9, pp. 66-71, 2013.

[12] Michalewicz Z. and Schoenauer M., "Evolutionary Algorithms for Constrained Parameter Optimization Problems," *Evolutionary Computation*, vol. 4, no.1, pp. 1-32, 1996.

[13] Niu S., Ong S., and Nee A., "An Improved Intelligent Water Drops Algorithm for Achieving Optimal Job-Shop Scheduling Solutions," *International Journal of Production Research*," vol. 50, no. 15, pp. 1-14, 2012.

[14] Niu S., Ong S., and Nee A., "An Improved Intelligent Water Drops Algorithm for Solving Multi-Objective Job Scheduling," *Engineering Applications of Artificial Intelligence*, vol. 26, no. 10, pp. 2431-2442 2013.
[15] Song X., Li B., and Yang H., "Improved Ant Colony Algorithm and its Applications in TSP," *in Proceedings of the 6th International Conference on Intelligent Systems Design and Applications*, Jinan, pp. 1145-1148.

Soarnapandy Selvarani received her BE degree in Computer Science and Engineering from Bharathiar University in 1991, and ME degree in Computer Science and Engineering from Manonmaniam Sundaranar University in 2004. Currently, She is working toward the PhD degree in Computer Science and Engineering under Anna University, Chennai, India. Her research work is in the area of grid and cloud computing with techniques for resource management and task scheduling.

Gangadharan Sadhasivam is working as a Professor in Department of Computer Science and Engineering in PSG College of Technology, Coimbatore, India. Her areas of interest include distributed computing, distributed object technology, grid and cloud computing. She has published 20 research papers in referred Journals and 32 papers in National and International Conferences. She has coordinated two AICTE-RPS projects in Distributed and Grid computing areas. She is also the coordinator for PSG-Yahoo research on Grid and Cloud computing.