SWARM INTELLIGENCE ALGORITHMS FOR OPTIMISATION: A SURVEY

Prof. S R Swamy[1], Ashish V Shenoy[2] , Aditya K S[3], Ashwin S L[4], Balachander R[5]
Department of Computer Science and Engineering,
RV College Of Engineering, Bangalore
mailtoswamy@rediffmail.com[1], ashishvshenoy@gmail.com[2], adi.chills@gmail.com[3], ashwin.lg341@gmail.com[4], dragontransformer@gmail.com[5]

***Abstract*— Swarm intelligence (SI) is the collective behavior of decentralized, self-organized systems, natural or artificial. This concept finds its applicability in artificial intelligence. The term swarm intelligence was coined in 1989 by Gerardo Beni and Jing Wang, in the context of cellular robotic systems. In this paper we have introduced 3 algorithms which have been inspired from natural world swarms namely Ant Colony Optimization algorithm, Particle Swarm Optimization algorithm and Intelligent Water Drop algorithm, along with their strengths, weaknesses and application domain.**

***Keywords*— optimization, swarm intelligence, intelligent water drops, ant colony optimization, particle search**

I.INTRODUCTION

Swarm intelligence is a modern interdisciplinary field of research. Algorithms belonging to this domain have been inspired from the collective behavior of a group of social insects such as bees, wasps and fireflies. These insects although have limited individual capability, when working as a group can perform quite a few complex tasks which are quite essential for their survival. Trivial problems such as foraging and storing food, travelling and picking up materials back to their nests require a particular type of planning which are solved by these swarms or insect colonies.
Here the individual behavior of these creatures (bees, wasps, ant, birds) is not important , it is the social behavior of these creatures which is of paramount significance. And this collective and social behavior of living creatures fascinated researchers and inspired them to undertake the study of today what is known as *Swarm Intelligence*. The efforts to imitate such behaviors through computer simulation finally resulted into the field of SI.
SI systems are mainly made up of a population of simple agents (an entity capable of performing/executing certain operations) interacting locally with one another and with their environment. Although there is normally no centralized control structure governing how individual agents should behave, interactions between such agents locally often lead to the emergence of global behavior. Many biological creatures such as fish schools and bird flocks clearly display structural order, the behavior of these organisms is integrated in such a way that even though they may change shape and direction, they appear to move as a single coherent entity. Let us take the example of a flock of birds to analyze the main properties of collective behavior[1],

1. ***Homogeneity***: every bird in flock has the same behavioral model. The flock moves without a leader, even though temporary leaders seem to appear.

2. ***Locality***: a bird's nearest flock-mates only influence its motion. Vision is considered to be the most important senses for flock organization.

3. ***Collision Avoidance***: individuals avoid colliding with nearby flock mates.

4. ***Velocity Matching***: they also attempt to match velocity with nearby flock mates.

5. ***Flock Centering***: they attempt to stay close to nearby flock mates and attempt to maintain a minimum distance between themselves and others at all times. This rule is given the highest priority and corresponds to a frequently observed behavior of animals in nature. If individuals are not performing an avoidance maneuver they tend to be attracted towards other individuals and to align themselves with neighbors.

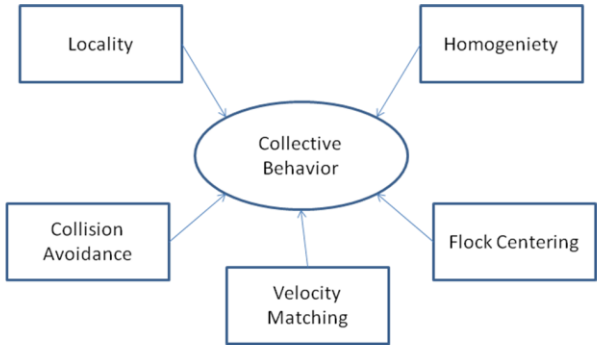

Figure 1

Also researchers have found that there are four different types of dynamic collective behavior of these creatures shown in Figure 2[1]:

1. ***Swarm***: an aggregate with cohesion, but a low level of polarization (parallel alignment) among members

2. ***Torus***: individuals perpetually rotate around an empty core (milling). The direction of rotation is random.

3. ***Dynamic parallel group***: the individuals are polarized and move as a coherent group, but individuals can move throughout the group and density and group form can fluctuate.

4. ***Highly parallel group***: much more static in terms of exchange of spatial positions within the group than the dynamic parallel group and the variation in density and form is minimal.

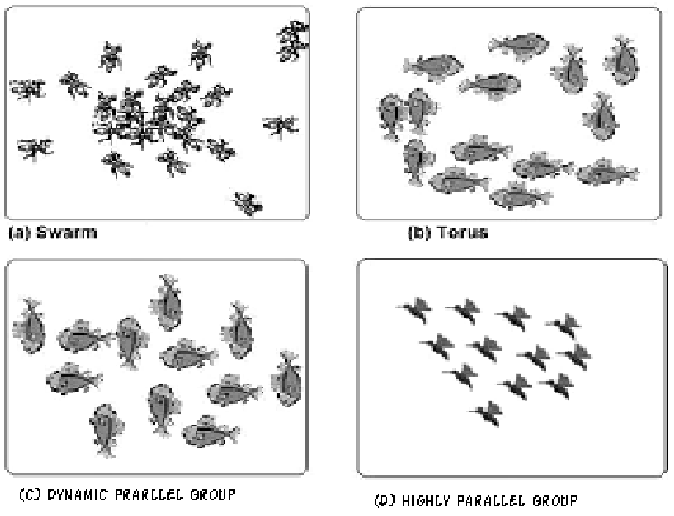

Figure 2[1]

Optimization

In mathematics, computer science and economics, **optimization**, or **mathematical programming**, is a process of choosing the best element from some set of available alternatives. In simplest case this means solving problems in which we try to minimize or maximize a real function by choosing the values of real or integer variables from within an allowed or given set.
More generally it refers to finding the “best available” values of a given system or a problem or a function.
Optimization problems generally have a huge number of local optimal points from which we have to choose the global optima.

II. ANT COLONY OPTIMISATION ALGORITHM

In the natural world, ants (initially) move around randomly, and upon finding food return to their colony leaving behind ***pheromone*** trails. When other ants find such a path traversed by a predecessor, they are likely to follow this trail instead of travelling randomly, returning and reinforcing it if they eventually find food.
Eventually, the pheromone trail starts to evaporate, thus reducing its attractivity. The longer it takes for an ant to travel down the path and back again, the longer the pheromones have to evaporate. A short path, by comparison, gets marched over more frequently, and thus the pheromone density becomes higher on shorter paths than longer ones. Pheromone evaporation ensures there is no convergence to a locally optimal solution. Suppose there was no evaporation at all, the paths chosen by the first few ants would tend to be more attractive to the following ones and it would lead to confusion due to the possibility of having more than one paths. In that case, the exploration of the solution space would be constrained. Hence pheromone evaporation is a key property in Ant Colony Optimization.
Therefore when one ant finds a good (i.e., short) path from the colony to a food source, other ants are more likely to follow that path, and ***positive feedback*** by these ants eventually leads all the ants following a single path which is the shortest among the path explored by other ants. The ant colony optimization imitates this behavior with "simulated ants" traversing around the graph of the problem to be solved.

A model explaining this behavior is as follows:[5]

1. An ant runs at random around the colony looking for food.
2. If it finds a food source, it returns more or less directly to the nest, leaving in its path a trail of pheromone
3. These pheromones are attractive to nearby ants and they will be inclined to follow, more or less directly, this track
4. As more and more ants travel the route they will strengthen the route
5. If there are two routes to reach the same food source then, after a certain amount of time, the shorter one will be marched upon by more ants than the long route
6. The short route will be increasingly enhanced, and therefore become more attractive
7. The long route will eventually vanish because pheromones are volatile

Eventually, all the ants have determined the optimal path and therefore have "chosen" the shortest route among a given set of routes. The Figure 3[5] gives a generalized overview of the path. Here N is nest and F is the food source. ‘a’ refers to the onward journey and ‘b’ refers to the return journey.

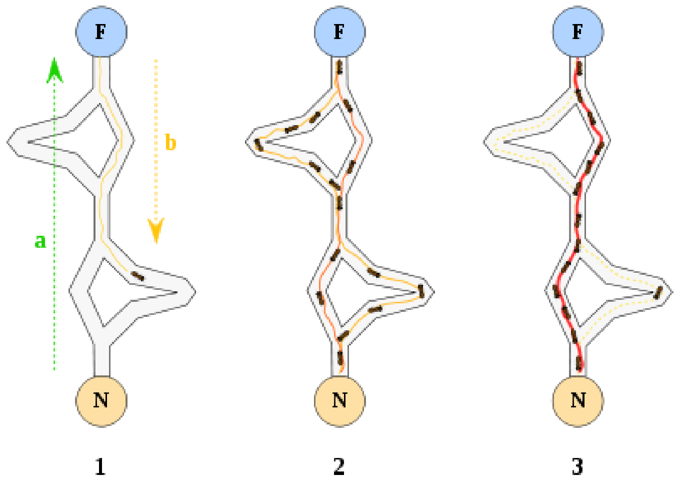

Figure 3[5]

Algorithm | Ant Colony Optimization Algorithm

```
1. Initialize pheromones
2. for each iteration do
3.    for k = 1 to number of ants do
          set out ant k at start node
4.        while ant k has not build a solution do
              choose the next node of the path
5.        end while
6.    end for
7.    update pheromones
8. end for
9. return best solution found
```

The intermediate solutions are referred to as solution states. Each ant moves from a state x to state y, corresponding to a more complete intermediate solution at each iteration of the algorithm[6]. Thus, each ant k computes a set $A_k(x)$ of feasible expansions to its current state in each iteration, and moves to one of these in probability. For ant k, the probability p_{xy}^{k} of moving from state x to state y depends on the combination of two values, which are the *attractiveness* η_{xy} of the move, as computed by some heuristic indicating *a prior* desirability of that move and the *trail level* τ_{xy} of the move, indicating how proficient it has been in the past to make that particular move. A posteriori indication of the desirability of that move is represented by the *trail level*. Trails are updated usually when all ants have completed their solution, increasing or decreasing the level of trails corresponding to moves that were part of "good" or "bad" solutions, respectively.
In general, the kth ant moves from state x to state y with probability [5]

$$p_{xy}^{k} = \frac{(\tau_{xy}^{\alpha})(\eta_{xy}^{\beta})}{\sum(\tau_{xy}^{\alpha})(\eta_{xy}^{\beta})}$$

where

τ_{xy} is the amount of pheromone deposited for transition from state x to y, $0 \leq \alpha$ is a parameter to control the influence of τ_{xy}, η_{xy} is the desirability of state transition xy (*apriori* knowledge, typically $1 / d_{xy}$, where d is the distance) and $\beta \leq 1$ is a parameter to control the influence of η_{xy}.

Strengths

- The ant colony algorithm can be run continuously and it adapts itself to changes in real time and hence are more advantageous than simulated annealing and generic algorithms in situations where the problems change dynamically.
- Pheromone evaporation is advantageous in avoiding the convergence to a locally optimal solution

Weakness

- Mainly applicable for discrete problems
- Theoretical analysis is difficult

Application Domain

- ACO is applied in routing problems such as
  --Travelling salesman
  --Vehicle routing
- ACO finds its application in solving scheduling problems such as
  --Job shop
  --Project scheduling
- It can be applied to solve Multiple Knapsack Problem
- It is used in set covering problems.
- It is used to solve Set Partition Problems
- It is instrumental in telecommunication networks
  -- Optimal network routing

More recently, an ACO algorithm designed for the challenging class of mobile ad hoc networks was shown to be competitive with state-of the-art routing algorithms, while at the same time offering better scalability.

Finally a mention of various other algorithms which are extensions or variations of the simple ACO algorithm shown above has been made

- Ant system (AS)
- Elitist Ant system (EAS)
- Rank Based Ant system (AS_{rank})
- Min-Max Ant system (MMAS)
- Ant Colony System(ACS)

III. PARTICLE SWARM OPTIMISATION ALGORITHM

Particle swarm optimization (PSO) is a stochastic optimization technique based on population and was developed by Dr. Eberhart and Dr. Kennedy in 1995. It was inspired by social behavior of bird flocking or fish schooling. The system is first initialized with a population of random solutions and searches for optima by updating generations [11]. This technique shares many similarities with evolutionary computation techniques such as Genetic Algorithms (GA). But, unlike GA, PSO has no evolution operators such as crossover and mutation. In PSO, the potential solutions, called particles, travel through the problem space by following the current optimum particles.

Each particle in the system keeps track of its coordinates in the problem space which are associated with the best solution (fitness) it has achieved so far. This value is called *pbest*. It also keeps track of the best value achieved by its neighbors. This value is called *lbest*. When a particle takes all the population as its topological neighbors, the best value is a global best and is called *gbest*.

The particle swarm optimization concept consists of the particles changing their velocity at each step toward its pbest and lbest locations. Acceleration is weighted by a random term, with separate random numbers being generated for acceleration toward *pbest* and *lbest* locations.

Consider the following situation: [12]

A group of birds are randomly searching food in an area. There is only one piece of food in the area being searched. All the birds do not know where the food is. But they know how far the food is in each iteration. So what's the best strategy to find the food? The effective one is to follow the bird which is nearest to the food.

PSO picked up from this scenario and used it to solve problems in optimization. In PSO, each single solution or entity is a "bird" in the search space. We call it "particle". All of particles have fitness values which are evaluated by the fitness function to be optimized, and have velocities which direct the flying of the particles. The particles travel through the problem space following the optimal particles.

PSO is initialized with a group of random particles or solutions and then searches for optima by updating generations.

- In every iteration, each particle is updated by following two "best" values.
- The first one is the best solution (fitness) it has achieved so far. (The fitness value is also stored.) This value is called pbest.
- Another "best" value that is tracked by the particle swarm optimizer is the best value, obtained so far by any particle in the population. This best value is a global best and called gbest.
- When a particle takes part of the population as its topological neighbors, the best value is a local best and is called lbest.
- After finding the two best values, the particle updates its velocity and positions with following equation
  v[ ] = v[ ] + c1 * rand( ) * (pbest[ ] - present[ ]) + c2 * rand( ) * (gbest[ ] - present[ ]) ---- (eqn a)
  present[ ] = present[ ] + v[ ] ---- (eqn b)
  v[ ] is the particle velocity
  present[ ] is the current particle (solution)
  pbest[ ] and gbest[ ] are defined as stated before
  rand ( ) is a random number between (0,1)
  c1, c2 are learning factors, usually c1 = c2 = 2.

Algorithm | Particle Swarm Optimization

```
1.For each particle
2.  Initialize particle
3.End
4. Do
5.  For each particle
6.      Calculate fitness value
7.      If the fitness value is better than the best fitness value
            (pBest) in history
8.          set current value as the new pBest
 9.     End For
10. Choose the particle with the best fitness value of all the
    particles as the gBest
 11.For each particle
       Calculate particle velocity according equation (a)
       Update particle position according equation (b)
    End For
12.While maximum iterations or minimum error criteria is not
       attained
    End While
```

Strengths

- PSO does not have genetic operators like crossover and mutation. Particles update themselves with the internal velocity.
- Compared to genetic algorithms (GAs), the information sharing mechanism in PSO is significantly different. In GAs it’s the chromosomes that share information with each other. So the whole population moves like a one group towards an optimal area. In PSO, only gBest (or lBest) gives out the information to others. Information is shared one-way only. The evolution only looks for the best solution.
- All the particles tend to converge to the best solution quickly even in the local version in most cases.
- Insensitive to scaling of design variables
- Simple implementation
- Easily parallelized for concurrent processing
- Very few algorithm parameters [10].

Weakness

- It converges slowly in refined search stage (which results in weak local search ability)
- This is computationally slower than Intelligent Water Drops Algorithm.

Application Domain

- Training of neural networks
  – Identification of Parkinson’s disease

– Extraction of rules from fuzzy networks
– Image recognition

- Optimization of electric power distribution networks
- Structural optimization
  – Optimal shape and sizing design
  – Topology optimization
- Process biochemistry
  --Optimizing Cancer therapy[10].

IV. INTELLIGENT WATER DROPS ALGORITHM

The rivers in nature have lots of twists and turns along their paths. This makes us wonder why these twists have been created and is there any logic or intelligence behind them? And if that is the case, can we use this mechanism that happens in rivers? Can we design and develop intelligent algorithms based on them? **Intelligent Water Drops** algorithm (IWD) is a swarm-based and nature-inspired optimization algorithm.

In IWD algorithm, IWDs are created with two important Properties [8]:

• velocity

• soil.

Both of these two properties may change during the lifetime of an IWD. An IWD flows from a source to a destination. The IWD starts its trip with an initial velocity and zero soil. During its journey, it travels through an environment from which it erodes some soil and also may gain some speed. The IWD velocity increases by an amount non-linearly proportional to the inverse of the soil between its current and next location. Therefore, a path with less soil makes the IWD become faster than a path with more soil. The amount of soil collected by the IWD is non-linearly proportional to the inverse of the time taken by the IWD to move from its current location to the next location. The time interval is calculated by the simple laws of physics for linear motion. Thus, the time taken is proportional to the velocity of the IWD and inversely proportional to the distance between the two locations. Moreover, those parts of the environment that are used with more IWDs will have less soil. The soil is the source material of information. The environment and the water drops both have memories for soil. An IWD needs a mechanism to choose the path to its next location or step. In this mechanism, the IWD gives preference for the paths having low soils over the paths having high soils. The method of path selection is implemented by imposing a uniform random distribution on the soils of the available paths. The probability of the next path to be selected is inversely proportional to the soils of the available paths.

Therefore, paths with lower soils have a higher chance to get selected by the IWD.

Algorithm| Intelligent Water Drop Optimization Algorithm

The algorithm of IWD is specified in the following steps: [8]

1. The graph (N, E) of the problem is given to the algorithm. The quality of the total-best solution T_{TB} is initially set to the worst value: $q(T_{TB}) = \infty$. The maximum number of iterations $iter_{max}$ is specified by the user. The iteration count $iter_{count}$ is set to zero. The number of water drops N_{IWD} is set to a positive integer value, which is usually set to the number of nodes N_c of the graph. For velocity updating, the parameters are a_v =1, b_v =0.01 and c_v = 1. For soil updating, a_s =1, b_s =0.01 and c_s = 1.
   The local soil updating parameter ρ_n=0.9, which is a small positive number less than one. The global soil updating parameter ρ_{IWD}=0.9, which is chosen from [0, 1]. Moreover, the initial soil on each path (edge) is denoted by the constant *InitSoil* such that the soil of the path between every two nodes *i* and *j* is set by soil(i, j) = InitSoil. The initial velocity of each IWD is set to InitV el. Both parameters InitSoil and InitV el are user selected and they should be tuned experimentally for the application.
2. Every IWD has a visited node list V_c(IWD), which is initially empty: V_c(IWD)= . Each IWDs velocity is set to InitV el. All IWDs are set to have zero amount of soil.
3. Spread the IWDs randomly on the nodes of the graph as their first visited nodes.
4. Update the visited node list of each IWD to include the nodes just visited.
5 Repeat Steps 5.1 to 5.4 for those IWDs with partial solutions.

5.1 For the IWD residing in node i, choose the next node j, which does not violate any constraints of the problem and is not in the visited node list vc(IWD) of the IWD, using the following probability P_i^{IWD}(j):

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

Such that

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

$$g(soil(i,j)) = \begin{cases} soil(i,j) & if \ \min\limits_{l \notin vc(IWD)} (soil(i,l)) \geq 0 \\ soil(i,j) - \min\limits_{l \notin vc(IWD)} (soil(i,l)) & else \end{cases}$$

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

5.2 For each IWD moving from node *i* to node *j*, update its velocity $vel^{IWD}(t)$ by

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

where $vel^{IWD}(t+1)$ is the updated velocity of the IWD.

5.3 For the IWD moving on the path from node *i* to *j*, compute the soil $\Delta soil(i, j)$ that the IWD loads from the path by

$$\Delta soil(i,j) = \frac{a_s}{b_s + c_s \, . \, time^2\left(i, j; vel^{IWD}(t+1)\right)} \quad (3)$$

such that

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

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 the IWD carries *soilIWD* by

$$soil(i,j) = (1-\rho_n) \, . \, soil(i,j) - \rho_n \, . \, \Delta soil(i,j)$$
$$soil^{IWD} = soil^{IWD} + \Delta soil(i,j) \quad (4)$$

6 Find the iteration-best solution T_{IB} from all the solutions T_{IWD} found by the IWDs using
$$T^{IB} = \arg \max_{\forall T^{IWD}} q(T^{IWD})$$
where function *q*(.) gives the quality of the solution.

7 Update the soils on the paths that form the current Iteration-best solution T_{IB} by
$$soil(i,j) = (1+\rho_{IWD}) \, . \, soil(i,j) - \rho_{IWD} \, . \, \frac{1}{(N_{IB}-1)} \, . \, soil^{IWD}_{IB} \quad \forall (i,j) \in T^{IB} \quad (6)$$
where N_{IB} is the number of nodes in the solution T_{IB}.

8 Update the total best solution T_{TB} by the current iteration-best solution T_{IB} using
$$T^{TB} = \begin{cases} T^{TB} & if \;\; q(T^{TB}) \;\geq\; q(T^{IB}) \\ T^{IB} & otherwise \end{cases} \quad (7)$$

9 Increment the iteration number by

Itercount = *Itercount* +1. Then, go to Step 2 if

Itercount <*Iter*max .

10 The algorithm stops here with the total-best solution T_{TB}.

Strengths:

- IWD allows searches and changes to its environment.
- The IWD algorithm finds the optimal solution of any given problem with probability one. (if the number of iterations *M* is sufficiently large)
- Numerical results show that IWD converged to better solutions in comparison to PSO, GA with less computational effort.
- It is robust and has fast convergence compared to PSO, GA.

Weakness

- Since the algorithm consists of heavy iterations it is complex and non intuitive.

Applications:

- The IWD algorithm is tested to find solutions of the n-queen puzzle with a simple local heuristic.
- IWD algorithm is tested for multiple (multidimensional) knapsack problems (MKP) in which near-optimal or optimal solutions are obtained.
- IWD has also been used for automatic multilevel thresholding of grey-level images.
- Intelligent Water Drop Algorithm is used for solving Economic Load Dispatch Problem. [9]
- IWD is used in UCAV (Unmanned Combat Aerial Vehicle) smooth trajectory planning.

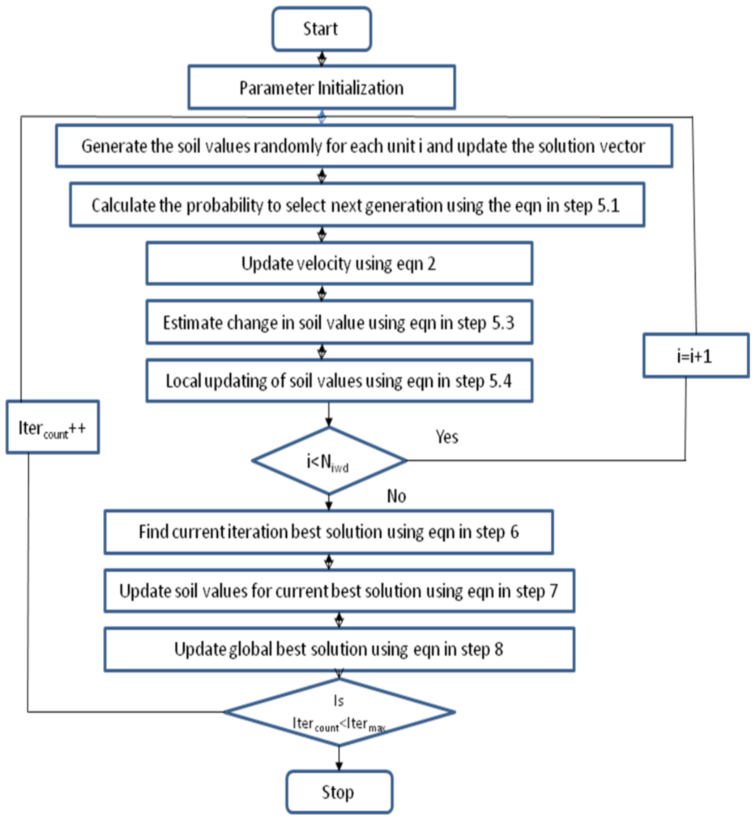
Start
Parameter Initialization
Generate the soil values randomly for each unit i and update the solution vector
Calculate the probability to select next generation using the eqn in step 5.1
Update velocity using eqn 2
Estimate change in soil value using eqn in step 5.3
Local updating of soil values using eqn in step 5.4
i=i+1
Iter count++
i<N iwd
Yes
No
Find current iteration best solution using eqn in step 6
Update soil values for current best solution using eqn in step 7
Update global best solution using eqn in step 8
Is Iter count<Iter max
Stop

Figure 4

V. OUTLOOK AND CONCLUSIONS

The main intention of this study is to get an overview and applicability of swarm intelligence algorithms for optimization. This paper tries to show how the behavior of even the tiniest of insects have contributed to solving most complex of problems in computer science such as TSP and routing. The fundamentals of the algorithm, that is how they are extracted from observing collective behavior of insects, flow of water drop in case of IWD were described along with their strengths and limitations. It can be concluded that ACO adapts itself to changes in real time and hence finds its application in ad hoc networks. The Ant colony Optimization, Particle Swarm Optimization and Intelligent Water Drops Optimization although were result of recent research and have vast applications and potential of becoming the most used algorithms in time. Nature truly is the best teacher.

VI. REFERENCES

[1]. *Swarm Intelligence Algorithms for Data Clustering* by Ajith Abraham, Swagatam Das, and Sandip Roy, Center of Excellence for Quantifiable Quality of Service (Q2S), Norwegian
University of Science and Technology, Trondheim, Norway.
[2]. *Fundamentals of Computational Swarm Intelligence* by Andries Engelbrecht. Wiley & Sons.
[3]. M. Dorigo, M. Birattari & T. Stützle, 2006 *Ant Colony Optimization: Artificial Ants as a Computational Intelligence Technique*
[4]. X Hu, J Zhang, and Y Li (2008). *Orthogonal methods based ant colony search for solving continuous optimization*

problems. Journal of Computer Science and Technology, pp.1-2.

[5]. *Wikipedia*, the free encyclopedia

[6]. Kennedy, J.; Eberhart, R. (1995). *"Particle Swarm Optimization" Proceedings of IEEE International Conference on Neural Networks*.

[7]. http://www.aco-metaheuristic.org/

[8]. *The intelligent water drops algorithm: a nature-inspired swarm-based optimization algorithm* by Hamed Shah-Hosseini, pp 1-9

[9]. *An Intelligent Water Drop Algorithm for Solving Economic Load Dispatch Problem by* S. Rao Rayapudi

[10].www.mae.ufl.edu/haftka/stropt/Lectures/**PSO_introduction**.pdf

[11]. www.swarmintelligence.org/tutorials.php

[12].*Particle Swarm Optimization for HW/SW Partitioning* by M B Abdelhalim and S E D Habib Cairo University, Egypt