Unit 4: Swarm Intelligence- What is swarm intelligence? Various animal behavior which have been used as examples, ant colony optimization, swarm intelligence in bees, flocks of birds, shoals of fish, ant-based routing, particle swarm optimization

The process of finding optimal values for the specific parameters of a given system to fulfill all design requirements while considering the lowest possible cost is referred to as an **optimization**. Optimization problems can be found in all fields of science.
Conventional optimization algorithms (Deterministic algorithms) have some limitations such as:

- single-based solutions
- converging to local optima
- unknown search space issues
-

To overcome these limitations, many scholars and researchers have developed several metaheuristics to address complex/unsolved optimization problems. Example: Particle Swarm Optimization, Grey wolf optimization, Ant colony Optimization, Genetic Algorithms, Cuckoo search algorithm, etc.

Swarm Intelligence

Swarm intelligence is an important concept in artificial intelligence and computer science with emergent properties. The essential idea of swarm intelligence algorithms is to employ many simple agents applying almost no rule which in turn leads to an emergent global behavior.

It was first proposed by **Kenedy and Russel Eberhart**.
As per definitions in literature, **Swarm Intelligence** means the implementation of collective intelligence shown by the behaviour of groups of simple agents like birds ,fish, ant. This was inspired by some collective behaviour of organisms like ants, wasps, bees,etc. It consists of simple organisms or agents that are interacting locally with each other and with environment.

As an individual, they are not intelligent but their intelligence lies in their ability to act in a coordinated way without the presence of any coordinator. These agents follow very simple rules and without any centralized control ,the interactions among these agents lead to the emergence of “intelligence”. Algorithms based on swarm intelligence are among the most popular. Many of them are ant colony optimization, particle swarm optimization , ABC ,cuckoo search , bat algorithm, firefly algorithm etc.

General principles

To model the broad behaviors arisen from a swarm, we introduce several general principles for swarm intelligence

Proximity principle: The basic units of a swarm should be capable of simple computation related to its surrounding environment. Here computation is regarded as a direct behavioral response to environmental variance, such as those triggered by interactions among agents. Depending on the complexity of agents involved, responses may vary greatly. However, some fundamental behaviors are shared, such as living-resource searching and nest building.

Quality principle: Apart from basic computation ability, a swarm should be able to response to quality factors, such as food and safety.

Principle of diverse response: Resources should not be concentrated in narrow region. The distribution should be designed so that each agent will be maximally protected facing environmental fluctuations.

Principle of stability and adaptability: Swarms are expected to adapt environmental fluctuations without rapidly changing modes since mode changing costs energy.

1. Ant Colony optimization (ACO)

The most recognized example of swarm intelligence in real world is the ants. To search for food, ants will start out from their colony and move randomly in all directions. Once a ant find food, it returns to colony and leave a trail of chemical substances called pheromone along the path. Other ants can then detect pheromone and follow the same path. The interesting point is that how often is the path visit by ants is determined by the concen- tration of pheromone along the path[3]. Since pheromone will naturally evaporate over time, the length of the path is also a factor. Therefore, under all these considerations, a shorter path will be favored because ants following that path keep adding pheromone which makes the concentration strong enough to against evaporation. As a result, the shortest path from colony to foods emerges.

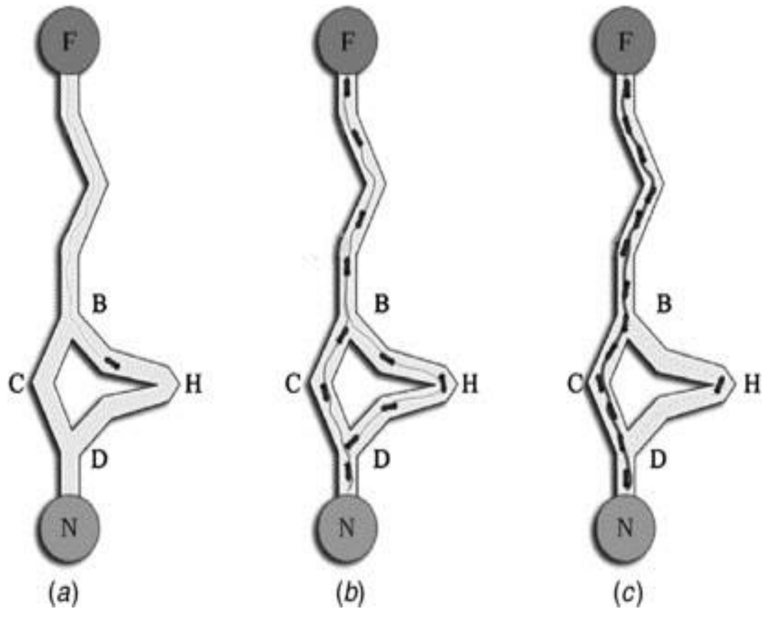

Figure 1: Ants select the shortest path

Learning from behaviors of real ants, a metaheuristic has been formulated for solving combinatorial problems, such as travelling salesman problem.

Define a combinatorial problem P as a triplet (S, Ω, f):

1. S is the search space over discrete decision variables $X_i \in D_i = \{v_i^1, \dots, v_i^{|D_i|}\}$.
2. Ω denotes the set of constraints.
3. Objective function $f : S \to \mathbb{R}$ to be maximized or minimized.

a solution $s \in S$ assigns values to variables that satisfies Ω and it asks for a solution $s^* \in S$ such that $f(s^*)$ is the global minimum or maximum.

The way that ant colony optimization algorithms tackle problems in this category is to employ the concept of pheromone. The ACO metaheuristic is split into three phases:

```
Initialization;
while not terminated do
    Construct solution using artifical ants;
    Local search (optional);
    Update pheromones;
end
```

Algorithm 1: The ACO metaheuristic

Solution construction Using m artificial ants, solutions $C = \{c_{ij}\}$, $i = 1 . . . n, j = i . . . D_i$ satisfies all the constraints Ω are constructed, where c_{ij} assigns the decision variable $X_i = v^{j}{}_{i}$. This can be also viewed as a random walk of ants on the construction graph $G_c(V, E)$.

Local search With specific design for individual problem, a local search could improve the constructed solution. However, since this is highly variable according to problems,local search is an optional process.

Update pheromones Pheromone values for promising solutions will be increased and values for undesired solutions will be decreased by pheromone evaporation. Thus thebest solutions will be rewarded with the highest concentration of pheromones.

Many NP-hard problems in computer science, which are problems with exponential time worst case complexity, can be solved using ACO algorithms, such as the assignment problem category and the scheduling problem category. There are proofs that ACOalgorithms will converge to these best-performing algorithms. However, the speed of convergence is unknown and the performance of ACO algorithms largely depend on if an optimal local search procedure can be found and this is very problem-specific.

ACO was inspired by the foraging behaviour of ants. They have a very strong ability to find the shortest path between their food and nest. While moving in the search of food they keep on releasing a chemical substance called pheromone on the path thereby indicating some favourable path for other members of ant colony. This indirect communication between the ants through pheromone quantity is called as stigmergy.The probability of selecting a path increases with the increase of pheromone in the path. Then, ants choose the path to follow by a probabilistic decision biased by the amount of pheromone: the stronger the pheromone trail, the higher its desirability. As they are depositing the pheromone on their path so their such behaviour results into a self-reinforcing process leading to the formation of paths marked by high pheromone concentration. The first ant colony optimization called as **Ant system (AS)** was proposed by **Dorigo** in early **1990**s and has been formalized into metaheuristic for optimization problem by Dorigo and his coworkers.

In ant colony optimization, the problem is tackled by simulating a number of artificial ants moving on a graph that encodes the problem itself: each vertex represents a city and each edge represents a connection between two cities. A variable called pheromone is associated with each edge and can be read and modified by ants. Ant colony optimization is an iterative algorithm. At each iteration, a number of artificial ants are considered. Each of them build a solution by walking from vertex to vertex on the graph with the constraint of not visiting any vertex that she has already visited in her walk. At each step of the solution construction, an ant selects the following vertex to be visited according to a stochastic mechanism that is biased by the pheromone: when in vertex i, the following vertex is selected stochastically among the previously unvisited ones. In particular, if j has not been previously visited, it can be selected with a probability that is proportional to the pheromone associated with edge (i, j). At the end of an iteration, on the basis of the quality of the solutions constructed by the ants, the pheromone values are modified in order to bias ants in future iterations to constructed. During the generations, the ACO uses the vaporization and splashing operators to avoid local minimum. Through the vaporization operator, the amount of pheromones is randomly decreased in the paths to avoid the trapping of the algorithm in local minima, while a splashing operator performs inversely. In the searching process of the ACO algorithm, the searching space will reduce gradually as the generation increases.

The original ACO is called as Ant System(AS), is structured into three main functions as follows: *AntSolutionsConstruct:* This function performs the solution construction process where the artificial ants move through adjacent states of a problem according to a transition rule, iteratively building solutions.

*Pheromone Update***:** performs pheromone trail updates. This may involve updating the pheromone trails once complete solutions have been built, or updating after each iteration. In addition to pheromone trail reinforcement, ACO also includes pheromone trail evaporation. Evaporation of the pheromone trials helps ants to forget‘ bad solutions that were learned early in the algorithm run.

Further an improvement **MAX-MIN Ant System (MMAS) (Stutzle and Hoos, 1997)** over the main ant system (AS) was proposed. In this, the main characteristic is that the pheromones are updated by the best ant only and the pheromone is bounded as maximum and min value as

$$\tau_{ij} \leftarrow \left[(1-\rho)\cdot\tau_{ij} + \Delta\tau_{ij}^{\text{best}}\right]_{\tau_{min}}^{\tau_{max}} ,$$

Where Δ is given by
$$\Delta\tau_{ij}^{\text{best}} = \begin{cases} 1/L_{\text{best}} & \text{if } (i,j) \text{ belongs to the best tour,} \\ 0 & \text{otherwise.} \end{cases}$$

While in case of AS,at each iteration,the pheromones are updated by all the ants that have built a solution in the iteration itself. The pheromone associated with edge i to edge j of the city for m no. of ants, as [28]

$$\tau_{ij} \leftarrow (1-\rho)\cdot\tau_{ij} + \sum_{k=1}^{m}\Delta\tau_{ij}^{k} ,$$

Where Δ is the quantity of pheromone laid at the edges.

$$\Delta\tau_{ij}^{k} = \begin{cases} Q/L_k & \text{if ant } k \text{ used edge } (i,j) \text{ in its tour,} \\ 0 & \text{otherwise,} \end{cases}$$

Another alternative approach, called the **ant colony system (ACS)** was introduced by **Dorigo and Gambardella (1997)** to improve the performance of ant system. The most interesting contribution of ACS is the introduction of a *local pheromone update* in addition to the pheromone update performed at the end of the construction process (called *offline* pheromone update). The local pheromone update is performed by all the ants after each construction step.

Each ant applies it only to the last edge traversed .It is based on four modifications of ant system: a different transition rule, a different pheromone trail update rule, the use of local updates of pheromone trail to favour exploration, and the use of candidate list to restrict the choice. A set of cooperating agents called *ants* cooperate to find good solutions to TSPs.It basically consists of two modes, i.e., the forward and backward modes. In the forward mode, a population of ants construct solutions probabilistically based on existing pheromone trails. In the backward mode, the solution constructed, including the solution quality is used to update pheromone trails. After several iterations, the ants will converge into a near-optimum or optimum solution.

The main goal of the local update is to diversify the search performed by subsequent ants during an iteration by decreasing the pheromone concentration on the traversed edges, ants encourage subsequent ants to choose other edges and, hence, to produce different solutions. This makes it less likely that several ants produce identical solutions during one iteration. The offline pheromone update, similarly to MMAS, is applied at the end of each iteration by only one ant, which can be either the *iteration-best* or the *best-so-far*.

2. Bee Colony Optimization

Just like ants, bees have similar food collecting behaviors. Instead of pheromones, bees colony optimization algorithm relies on the foraging behavior of honey bees. At the first stage, some bees are sent out to look for promising food sources. After a good food source is located, bees return back to colony and perform a waggle dance to spread out information about the source. Three pieces of information are included: 1. distance, 2. direction, 3. quality of food source. The better the quality of food source, the more bees will be attracted. Therefore, the best food source emerges.

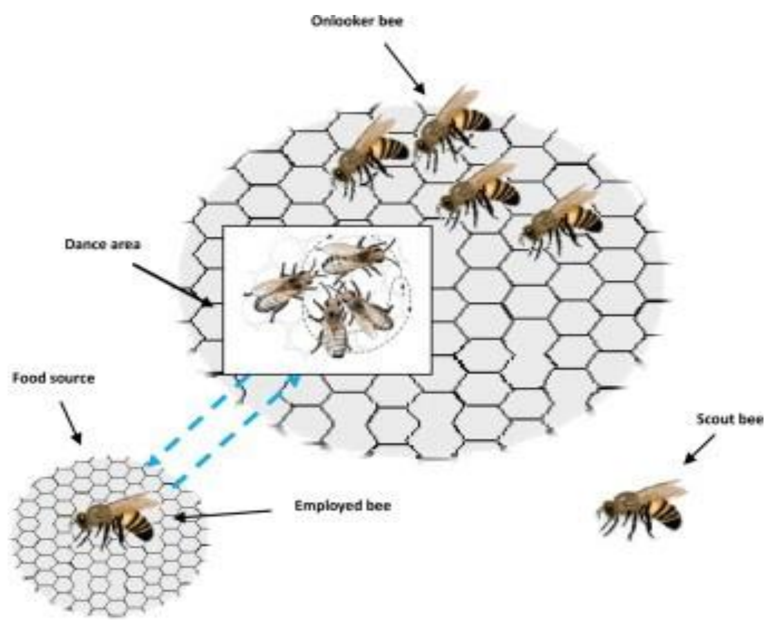

Figure 4.2: How bees work to find food sources

The metaheuristic extracted from the foraging behaviors of bees can also be applied to solve combinatorial problems; especially problems involve global minimum or maximum. Similarly, the BCO metaheuristic undergo several phases:

```
Initialization;
while not terminated do
    Employed Bees Phase;
    Onlooker Bees Phase;
    Scout Bees Phase;
    Memorize the best solution;
end
```

Algorithm 2: The BCO metaheuristic

Initialization All the food sources $\vec{F}_m$, $m = 1, \ldots, N$ are initialized. $\vec{F}_m$ are solutions to the optimization problems and will be tuned by BCO algorithm to minimize or maximize objective function f defined above.

Employed Bees Employed bees will search in the neighborhood of $\vec{F}_m$ from memory with a random vector $\vec{R}_m$. A fitness will be calculated to determine if $\vec{R}_m$ leads to a better food source. The usual choice for fitness function T is:

$$T(\vec{x_m}) = \begin{cases} \frac{1}{1+f(\vec{x_m})}, & \text{if } f(\vec{x_m}) \geq 0 \\ 1 + |f(\vec{x_m})|, & \text{if } f(\vec{x_m}) < 0 \end{cases}$$

Onlooker Bees After employed bees shared information about food sources, onlooker bees will probabilistically choose their destination accordingly. Usually, this is calculated depending on the fitness values provided by employed bees. For example, with the above defined fitness value $T(\vec{x}_m)$, the probability value p_m can be calculated:

$$p_m = \frac{T(\vec{x_m})}{\sum_{m=1}^{N} T(\vec{x_m})}$$

With more onlooker bees recruited to richer resources, positive feedback also arises for richer resources.

Scout bees The third kind of bees is the scout bees. They are usually these employed bees abandoned by the algorithms because the quality of food sources they found is poor. Scout bees will again start from the beginning and search for food sources randomly. However, negative feedback will lower the attractiveness of their previous found food sources.

The BCO algorithms have interesting applications in numerical optimizations, for example, it can be used to find global optimal solutions of functions. Moreover, re- cent studies suggest that the BCO algorithms can also be applied to problems in shop scheduling, neural network training and imaging processing.

3. ABC (Artificial bee colony)

The Artificial Bee Colony (ABC) algorithm is a population-based optimization algorithm inspired by the foraging behavior of honey bees. It was introduced by Karaboga in 2005 and has since been applied to various optimization problems, particularly in engineering, machine learning, and operations research. It was inspired by the intelligent behavior of the honey bees Such algorithms are classified into two; foraging behavior and mating behavior.

In ABC algorithm, the colony of artificial bees contains three groups of bees: *employed bees, onlookers and scouts.*
A bee waiting on the dance area for making a decision to choose a food source is called onlooker and one going to the food source visited by it before is named employed bee. The other kind of bee is scout bee that carries out random search for discovering new sources. The position of a food source represents a possible solution to the optimization problem and the nectar amount of a food source corresponds to the quality (fitness) of the associated solution. A swarm of virtual bees is generated and started to move randomly in two-dimensional search space. Bees interact when they find some target nectar and the solution of the problem is obtained from the intensity of these bee interactions. A randomly distributed initial population solutions (xi=1,2…D) is being dispread over the D dimensional problem space. An employed bee produces a modification on the position (solution) in her memory depending on the local information (visual information) and tests the nectar amount (fitness value) of the new source (new solution). Provided that the nectar amount of the new one is higher than that of the previous one, the bee memorizes the new position and forgets the old one. After all employed bees complete the search process; they share the nectar information of the food sources and their position information with the onlooker bees on the dance area.

In the next phase **Reproduction**, based on the probability value associated with the food source, **Pi**, the artificial onlooker bee chooses a food . In the last phase, **Replacement of bee and Selection**, *if* a position can not be improved further through a predetermined number of cycles, then that food source is assumed to be abandoned. The value of predetermined number of cycles is an important control parameter of the ABC algorithm, which is called ―*limit*‖ for abandonment. After each candidate source position is produced and then evaluated by the artificial bee, its performance is compared with that of its old one. If the new food has an equal or better nectar than the old source, it is replaces the old one in the memory. Otherwise, the old one is retained in the memory. The local search performance of ABC algorithm depends on neighborhood search and greedy selection mechanisms performed by employed and onlooker bees. The global search performance of the algorithm depends on random search process performed by scouts and neighbor solution production mechanism performed by employed and onlooker bees.

Here's a simplified explanation of how the ABC algorithm works:

Initialization: The ABC algorithm starts by randomly generating an initial population of candidate solutions, which are represented as "food sources" in the context of the bee colony analogy.

Employed Bees Phase: In this phase, employed bees explore the neighborhood of their current food sources. Each employed bee evaluates the quality of its food source by performing a local search around it. This local search can be performed using various optimization techniques, such as gradient descent or random perturbations.

Onlooker Bees Phase: Onlooker bees select food sources based on their quality (fitness) to exploit. The probability of selecting a food source is proportional to its fitness. Onlooker bees then perform local searches around the selected food sources to find better solutions.

Scout Bees Phase: Scout bees are responsible for discovering new food sources. They randomly explore the search space and replace the food sources that have been abandoned or have not improved for a certain number of iterations.

Update: After each iteration, the quality of the food sources is updated based on the fitness of the solutions. The algorithm iterates through these phases until a stopping criterion is met, such as reaching a maximum number of iterations or finding a satisfactory solution.

The ABC algorithm is characterized by its simplicity, efficiency, and ability to handle optimization problems with high-dimensional search spaces. However, its performance may vary depending on the problem being solved and the choice of algorithm parameters, such as the number of bees, the neighborhood size, and the abandonment threshold.

Overall, the ABC algorithm is a versatile optimization technique that has been successfully applied to a wide range of real-world problems, including parameter optimization, feature selection, clustering, and scheduling.

The main steps of the algorithm are given below:

- Initial food sources are produced for all employed bees
- REPEAT
    - Each employed bee goes to a food source in her memory and determines a closest source, then evaluates its nectar amount and dances in the hive
    - Each onlooker watches the dance of employed bees and chooses one of their sources depending on the dances, and then goes to that source. After choosing a neighbour around that, she evaluates its nectar amount.
    - Abandoned food sources are determined and are replaced with the new food sources discovered by scouts.
    - The best food source found so far is registered.
- UNTIL (requirements are met)

4. Fish Swarm algorithm (FSA):

The Fish Swarm Algorithm (FSA) is a population-based optimization algorithm inspired by the behavior of fish shoals. It was proposed by P.E. Runkler in 1997. FSA mimics the collective behavior of fish in their search for food, where individuals coordinate their movements to find the best feeding spots. This algorithm has been applied to solve various optimization problems in fields such as engineering, robotics, and data mining.

FSA presents a strong ability to avoid local minimums in order to achieve global optimization. A fish is represented by its D-dimensional position Xi = (x1, x2, . . ., xk, . . ., xD), and food satisfaction for the fish is represented as FSi. The relationship between two fish is denoted by their Euclidean distance dij = ||Xi −Xj||. FSA imitates three typical behaviors, defined as searching for food, swarming in response to a threat and following to increase the chance of achieving a successful result.

Searching is a random search adopted by fish in search of food, with a tendency towards food concentration. The objective is to minimize FS (food satisfaction). **Swarming:** aims in satisfying food intake needs, entertaining swarm members and attracting new swarm members. A fish located at Xi has neighbors within its visual. Xc identifies the center position of those neighbors and is used to describe the attributes of the entire neighboring swarm. If the swarm center has greater concentration of food than is available at the fish‘s current position Xi (i.e., FSc < FSi), and if the swarm (Xc) is not overly crowded (ns/n < δ), the fish will move from Xi to next Xi+1, toward Xc .

Following behavior implies when a fish locates food, neighboring individuals follow. Within a fish's visual, certain fish will be perceived as finding a greater amount of food than others, and this fish will naturally try to follow the best one(Xmin) in order to increase satisfaction(i.e., gain relatively more food[FSmin < FSi] and less crowding[nf/n < δ]). nf represents number of fish within the visual of Xmin. Three major **parameters** involved in FSA include visual distance (visual), maximum step length (step), and a crowd factor. FSA effectiveness seems primarily influenced by the former two (visual and step).

Here's a simplified explanation of how the Fish Swarm Algorithm works:

Initialization: The algorithm starts by randomly generating an initial population of fish, each representing a potential solution to the optimization problem.

Evaluation: The fitness of each fish is evaluated based on its performance in the problem domain. This could involve evaluating a cost function, objective function, or any other measure of solution quality.

Movement: Fish in the population adjust their positions in the search space based on local interactions with neighboring fish. This movement is guided by both individual behavior and collective behavior, as follows:

Individual Movement: Each fish explores its local environment by adjusting its position randomly or based on a heuristic. This allows fish to explore nearby regions of the search space.

Collective Movement: Fish are attracted to regions of high fitness (good solutions) and repelled from regions of low fitness (poor solutions). This collective behavior helps the swarm converge towards better solutions.

Update: After each iteration, the fitness of the fish is reassessed, and their positions are updated based on the evaluation results and the movement rules.

Termination: The algorithm continues iterating until a termination criterion is met, such as reaching a maximum number of iterations or finding a satisfactory solution.

Yun Cai ,2010 proposed Artificial Fish School Algorithm Applied in a Combinatorial Optimization Problem for a berth allocation problem (BAP) and showed that the algorithm has better convergence performance than genetic algorithm (GA) and ant colony optimization (ACO). **Ying Wu,2011** also proposed **Knowledge-based artificial fish-swarm algorithm**with crossover, **CAFAC**, is proposed to enhance the optimization efficiency and combat the blindness of the search of the AFA. There are two important parameter i.e visual and step.both are required to set initially and remain constant throughout till the termination of the algorithm.if theses parameters are selected as low then its capable of passing local optima and reach global optimum while in case of selecting lower values , it causes FSA to act in local searching. so **Reza Aziz,2014** proposed an **adaptive FSA(AFSA)**in which a new parameter, called movement Weight is introduced to adjust visual and step adaptively and consequently controls the movements of artificial fish towards the target.It is helpul in maintaining the equilibrium between global and local searches. FSA has been successfully applied for the gene rank aggregation problem.

The Fish Swarm Algorithm shares similarities with other population-based optimization algorithms such as Particle Swarm Optimization (PSO) and Genetic Algorithms (GA). However, its inspiration from fish shoaling behavior gives it a unique approach to exploration and exploitation in the search space.

One advantage of FSA is its ability to handle multimodal and dynamic optimization problems, where the landscape of the objective function may change over time. However, like other metaheuristic algorithms, the performance of FSA can be sensitive to its parameter settings and the choice of problem-specific encoding and fitness evaluation.

5. Particle Swarm Optimization

PSO is a population based optimization algo proposed by **Eberhart and Kenedy** in **1995**.This is actually the simulation of social behaviour of bird flocking or fish schooling. Birds follow the bird with the shortest distance to food in order to find position of the food. It consists of a swarm of particles and each particle flies through the multi-dimensional search space with a velocity and find its bets position after some iteration. At each of the iteration,the velocity of the particle is constantly updated by the particle's best position (pbest) and by best position of the particle's neighbours(gbest) to compute a new positon that the particle is to fly on. Each particle is a solution of the considered problem and uses its own experience and the experience of neighbour particles/solutions to choose how to move in the search space.

Steps in PSO are

1. Randomly initialize particle positions and velocities.

2. For each particle i:

-Evaluate fitness *yi* at current position *xi*

-If *yi* is better than *pbesti* then update *pbest i*and the current particle‘s position, xi, as *pi*

-If *yi* is better than *gbesti* then update *gbesti* and *gi*

3. For each particle

-Update velocity *vi* and position *xi* using standard PSO equations[21]

$$x_{k+1}^{i} = x_{k}^{i} + v_{k+1}^{i}$$

$$v_{k+1}^{i} = v_{k}^{i} + c_1 r_1 (p_{k}^{i} - x_{k}^{i}) + c_2 r_2 (p_{k}^{g} - x_{k}^{i})$$

Where x_k^i represents Particle position, v_k^i represents Particle velocity, p_k^i represents Best "remembered" position ,c1,c2 represents cognitive and social parameters, r1 ,r2 are random numbers between 0 and 1.

There were certain problems in original PSO. It can accelerate “collapse” of swarm for better local search – at the cost of higher possibility of premature convergence .Higher acceleration coefficients(r1,r2) result in less stable systems in which the velocity has a tendency to explode. So in addition to these accerlation coefficients, it is required to place a limit on the velocity too. To fix this, the velocity *vi* is usually kept within the range[-vmax, vmax].However, limiting the velocity does not necessarily prevent particles from leaving the search space, nor does it help to guarantee convergence.

Eberhart and Shi(1998) proposed an **adaptive PSO(APSO)** in which they introduced inertia weight ω into the previous PSO. Through adjusting ω ,the performance of PSO can be improved. It was introduced to control the velocity explosion thereby controlling the balance between exploration and exploitation. Eberhart and Shi suggested to decrease ω over time (typically from 0.9 to 0.4) and thereby gradually changing from an exploration to exploitation.

On a comparison with EA: Even though PSO is a good and fast search algorithm, it has its limitations while solving real world problems. The two standard mathematical PSO equations restrict additional heuristics related to the real-world problem to be incorporated in the algorithm, while in the case of EA, heuristics can be easily incorporated in the population generator and mutation operator to prevent wrong updates to the individuals to infeasible solutions. Therefore, PSO will not perform well in its search in complex multiconstrained solution spaces, which are the case for many complex real world problems like scheduling. To overcome these limitations of PSO, **Srinivasan and Tian Hou** proposed an algorithm **PSO-EA**, a hybridized evolutionary algorithm, which allows flexible incorporations of the real world heuristics into the algorithm, while retaining the workings of PSO.PS-EA is a hybrid model of EA and PSO. In this, the mathematical equations are replaced by self-updating mechanism (SUM). An additional elite particle popbest, which is the best particle of a current swarm, is introduced to SUM for faster convergence along with pbest and gbest.

PSO Algorithm

Parameters of problem:

- Number of dimensions (d)
- Lower bound (minx)
- Upper bound (maxx)

Hyperparameters of the algorithm:

- Number of particles (N)
- Maximum number of iterations (max_iter)
- Inertia (w)
- Cognition of particle (C1)
- Social influence of swarm (C2)

Algorithm:

```
Step1: Randomly initialize Swarm population of N particles Xi ( i=1, 2, …, n)
Step2: Select hyperparameter values
        w, c1 and c2
Step 3: For Iter in range(max_iter):  # loop max_iter times
```

```
            For i in range(N):  # for each particle:
               a. Compute new velocity of ith particle
                    swarm[i].velocity =
                         w*swarm[i].velocity +
                         r1*c1*(swarm[i].bestPos - swarm[i].position) +
                         r2*c2*( best_pos_swarm - swarm[i].position)
               b. Compute new position of ith particle using its new velocity
                    swarm[i].position += swarm[i].velocity
               c. If position is not in range [minx, maxx] then clip it
                    if swarm[i].position < minx:
                        swarm[i].position = minx
                    elif swarm[i].position > maxx:
                        swarm[i].position = maxx
               d. Update new best of this particle and new best of Swarm
                     if swaInsensitive to scaling of design variables.rm[i].fitness <
swarm[i].bestFitness:
                        swarm[i].bestFitness = swarm[i].fitness
                        swarm[i].bestPos = swarm[i].position

                     if swarm[i].fitness < best_fitness_swarm
                        best_fitness_swarm = swarm[i].fitness
                        best_pos_swarm = swarm[i].position
             End-for
         End -for
```

Step 4: Return best particle of Swarm

Advantages of PSO:

1. Insensitive to scaling of design variables.
2. Derivative free.
3. Very few algorithm parameters.
4. Very efficient global search algorithm.
5. Easily parallelized for concurrent processing.

Disadvantages of PSO:

1. Slow convergence in the refined search stage (Weak local search ability)