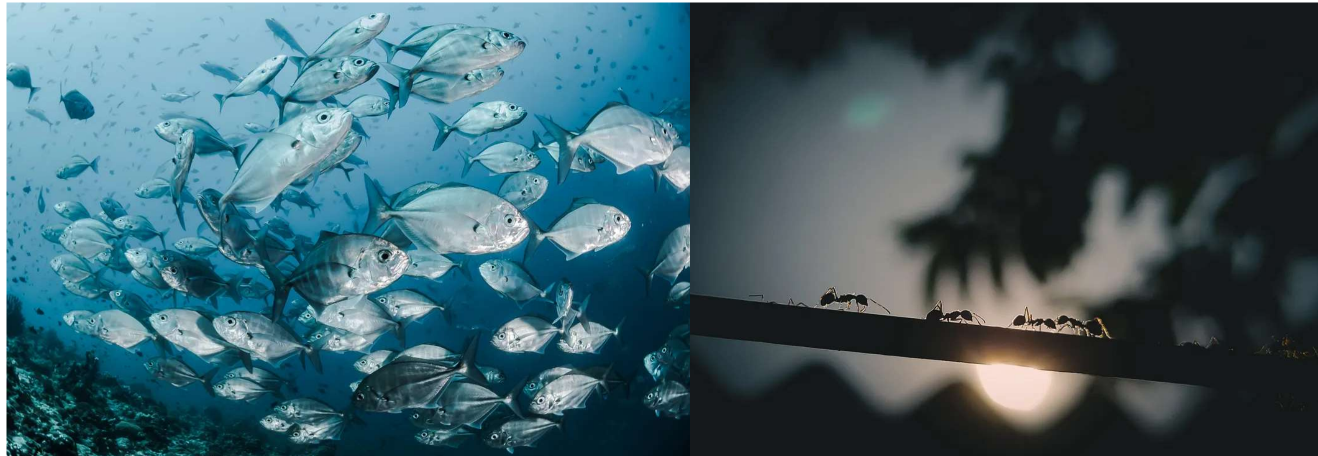
Swarm Intelligence Techniques

Arna Fariza – S2 TIK PENS



Introduction to Swarm Intelligence

- A swarm is a collection of agents or organisms; swarm intelligence can be defined as the social behaviours of a swarm in which autonomous individuals interact with each other in a decentralised and self-organised manner.
- The interaction of individuals improves the empirical knowledge about the environment and brings the swarm to the optimal state.

Introduction to Swarm Intelligence

- There are some nature-inspired algorithms that mimic swarm intelligence.
- Ant Colony Optimisation (ACO) is derived from ants.
- Artificial Bee Colony (ABC) is inspired by honeybees swarming around their hives.
- This article is about Particle Swarm Optimisation (PSO) which is hinted at by bird flocking and fish schooling.

Introduction to Swarm Intelligence

- For example, ants are known to find the shortest path from their colony to a food source.
- In the beginning, individuals explore various directions from and to the destination.
- When a favourable route is found, ants mark the path with pheromones which are chemical substances ants deposit on the ground.
- As more ants take the same trail, the pheromones intensify, attracting more ants.
- Consequently, the majority of ants follow and converge to the shortest path.

Particle Swarm Optimization (PSO)

- PSO was initially introduced by James Kennedy and Russell Eberhart in 1995.
- They hypothesised that the social sharing of the information would offer advantages to the swarm.
- Rather than competing for food, individual members of the fish school can profit from other members' discoveries and experiences whenever the resource is distributed beyond prediction.
- The key word is "social interaction". Social behaviour increases the capability of an individual to adapt; as a result, intelligence emerges in the swarm.
- The adaptability of individuals and collective intelligence are related to each other.

Particle Swarm Optimization (PSO)

- The algorithm of PSO is simple.
- Particles are a number of simple entities in a search space.
- We create a population of particles and measure their individual fitness with an objective function of the problem.
- Particles are then moved from their current to the next position based on their personal best location, and on the swarm's best location so far.
- By iterating the moves the swarm gradually reaches an optimal point of the objective function over generations.

Particle Swarm Optimization (PSO)

- Overview of PSO mechanics: particles representing solutions, velocity and position updates, cognitive and social components.

```
Number of particles          : i
Number of dimensions         : n
Fitness function             : f(x_i)
Particles                    : x_i = (x_i1, x_i2, ..., x_in)
Current velocity             : v_i = (v_i1, v_i2, ..., v_in)
Individual particle's best   : p_i = (p_i1, p_i2, ..., p_in)
Global particles' best       : p_g = (p_g1, p_g2, ..., p_gn)
Inertia component            : w * v_i(t)
Cognitive component          : c_1 * r_1 * (p_i - x_i(t))
Social component             : c_2 * r_2 * (g_i - x_i(t))
Velocity adjustment          : v_i(t+1) <- Inertia+Cognitive+Social
Position adjustment          : x_i(t+1) <- x_i(t)+v_i(t+1)
```

Particle Swarm Optimization (PSO)

- The velocity adjustment is influenced by 3 factors: the previous velocity (Inertia component), the individual particle's best position (Cognitive component) and the swarm's best positions (Social component).
- The velocity is a speed of a moving particle to a given direction.
- The particle's movement is affected by these weights in each direction.
- The coefficient w is called inertia weight which is a force to keep the particle moving in the same direction as the previous generation.
- c_1 and c_2 are constant acceleration values where $c_1=c_2$ is applied in the original algorithm
- r_1 and r_2 denote hyper parameters and they cause some random perturbations.
- The higher value of these parameter values results in a more responsive movement of the particles.
- We also assume that the fitness function is for the minimisation problem in our case.
- Therefore the individual particle's best position p_i is over-written by x_i when $f(x_i) < f(p_i)$.

PSO Algorithm

```
1. Initialise the particle population array x_i
2. Loop
3. For each particle, calculate the fitness using the
   fitness function f(x_i)
4. Compare the current fitness value with its best p_i.
   Replace the best with the current value x_i
   if it is better than the best.
5. Check the swarm’s best particle from individual particle’s best
   and assign the best array to the global best p_g.
6. Calculate the velocity v_i(t+1) and update the position of
   the particles to x_i(t+1)
7. If a criterion is met, exit the loop.
8. End loop
```

PSO Algorithm

- In the first generation, the particles are scattered.
- They quickly move towards the bottom of the grid, and it looks like the algorithm is working as expected.
- We could do some more experiments by changing the fitness functions and hyper-parameters like the inertia weight w and cognitive and social coefficients (c1 and c2) to optimise the algorithm.

Summary of PSO

- Swarm intelligence like PSO is a class of metaheuristics that is believed to find a near-optimal solution for complex optimisation problems with a reasonable computational time.
- It is especially useful if we apply the algorithm to train a neural network.
- The benefit is twofold: global search and parallelisation.

Summary of PSO

- We can translate each particle to be an n-dimensional array that represents the weights between neurons.
- Unlike the backpropagation learning algorithm that searches the optimum point locally, PSO makes it possible to explore many different sets of weight parameters simultaneously, thus helping to avoid the path to the local minima.
- Furthermore, the fitness function is independent of a network topology; the computation to evaluate each particle's fitness can be parallelised.

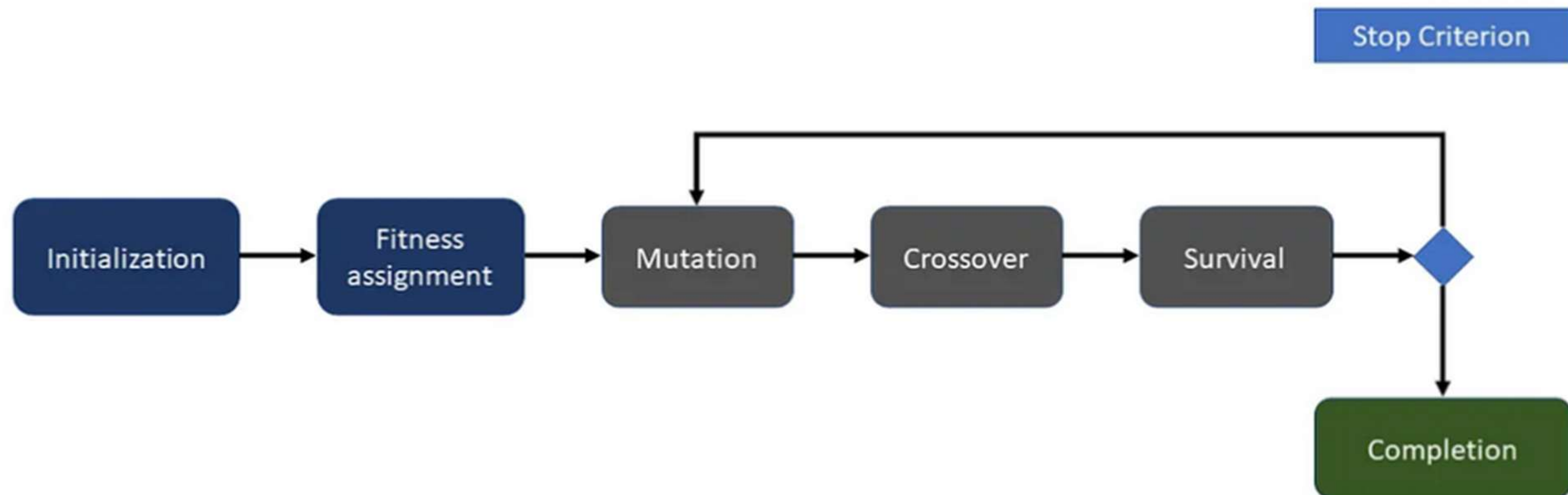
Differential Evolution (DE)

- Differential Evolution (DE) (Storn & Price, 1997) is an Evolutionary Algorithm (EA) originally designed for solving optimization problems over continuous domains.
- It has a simple implementation yet a great problem-solving quality, which makes it one of the most popular population-based algorithms, with several successful applications reported.

DE Requirement

1. Ability to handle non-differentiable, nonlinear, and multimodal cost functions.
2. Parallelizability to cope with computationally intensive cost functions.
3. Ease of use: few control variables to steer the minimization. These variables should also be robust and easy to choose.
4. Good convergence properties: consistent convergence to the global minimum in consecutive independent trials.

DE Basic Structure



DE Algorithm

- The algorithm starts by initializing a population based on a user-specified number of individuals N and the boundaries of each decision variable of the problem. Each individual corresponds to a vector of optimization variables. A choice of N between 5 and 10 times the number of decision variables might be a good start.
- Individuals are assigned a fitness value, based on their corresponding objective function values and possibly constraint values. Originally DE has no rule for constraint handling, which was later the focus of several articles. A useful approach was proposed by Lampinen (2002), as it has presented competitive performance and requires no additional control parameter when being implemented. It is the method adopted in the scipy DE implementation.

DE Algorithm

- The population then iterates through successive generations until some stopping criteria are met. In each of these iterations, new trial vectors are produced by operations known as mutation and crossover. The trial vectors are then compared to their corresponding parents of the same index and the best of each pair is passed to the next generation. Stopping criteria are usually based on improvements in the objective function and the number of generations.

DE Algorithm

- Several reproduction schemes have been proposed for DE. Usually, they are denoted DE/x/y/z, in which x corresponds to the mutation parent selection scheme, y to the number of difference vectors, and z to the crossover strategy.
- Probably the most popular *mutation scheme* is the DE/rand/1

$$\boldsymbol{v}_i = \boldsymbol{x}_{r1} + F(\boldsymbol{x}_{r2} - \boldsymbol{x}_{r3})$$

- In which, v corresponds to a mutant vector of index i, and r1, r2, and r3 are three indexes mutually different and different from i. The parameter F is a control parameter user-specified, denoted mutation parameter or scale factor.

DE Algorithm

- The most usual crossover strategy is the binomial crossover or just bin

$$u_{i,j} = \begin{cases} v_{i,j}, & rand(0,1)_{i,j} < CR \\ x_{i,j}, & rand(0,1)_{i,j} \geq CR \end{cases}$$

- In which u corresponds to a trial vector created combining elements of the corresponding mutant vectors v and target x. The parameter CR controls the probability of inheriting one attribute from each and an additional rule states that at least one attribute of u must be inherited from v to avoid duplicates.

Comparison of PSO and DE

- Key differences: convergence behavior, computational efficiency, parameter tuning.
- Highlight when to use one over the other.

Conclusion

- Hybrid Methods: Combining swarm intelligence with other AI techniques.
- Recent Advancements: Variants like quantum-behavior PSO or adaptive DE show ongoing evolution.