16 - Particle Swarm Optimization: algorithm and example

- Explores collectively the search space in order to find the optimal solution
- An individual will, at every iteration, choose his path based on his own best path, his current path, and the groups best path
- The hope is that the group of particles will find the best solution

ALGORITHM

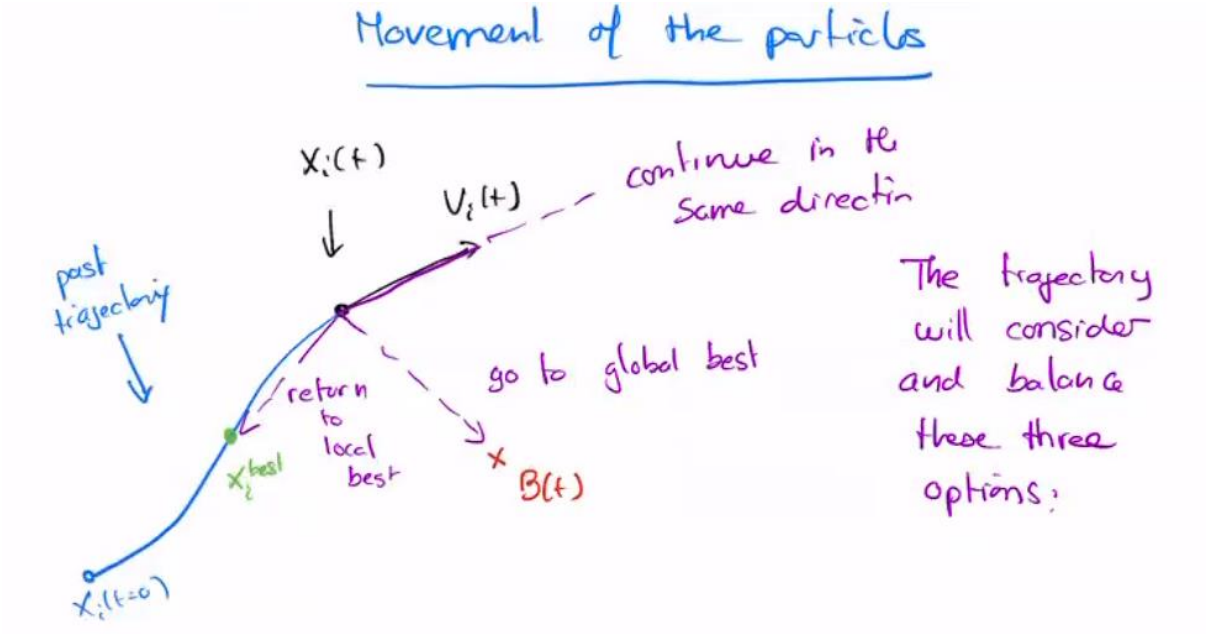
- ➔ It is a population metaheuristic (at each iteration the number of individuals stays constant)
- ➔ Xi(t) is the position of particle I at iteration t
- ➔ Each xi is a possible solution and it has a corresponding fitness
- ➔ Particles explore the search space because they have a velocity, which makes them move from iteration t to iteration t+1

At each iteration each particle will update its local best (his own best solution)

$$X_i^{best}(t) = \begin{cases} X_i(t) & \text{if } f(X_i(t)) \text{ is better than } f(X_i^{best}) \\ X_i^{best}(t-1) & \end{cases}$$

At each iteration the global best is also updated -> B(t)

$$B(t) = \underset{X_i^{best}}{\operatorname{argmax}}\ f(X_i^{best}) \quad \text{(for a maximization problem)}$$



current local best

$$\begin{cases} V_i(t+1) = \omega V_i(t) + c_1 r_1(t+1)\left[x_i^{best}(t) - x_i(t)\right] \\ \qquad\qquad + c_2 r_2(t+1)\left[B(t) - x_i(t)\right] \\ x_i(t+1) = x_i(t) + V_i(t+1) \end{cases}$$

Global

- ω is a parameter or $\omega < 1$ is an inertia parameter. How much of the previous velocity we will keep
- c_1 is called the cognitif coefficient as it takes into account the knowledge of the individual.
- c_2 is called the social coefficient as it considers the knowledge of the group. $c_1 = c_2 \cong 2$
- r_1 and r_2 are random numbers uniformly distributed in $[0, 1[$

- ➔ Everything is a vector!!
- ➔ Particles are placed randomly in S with v = 0
- ➔ We need to have search boundaries, many options are available, like a particle bouncing etc...
- ➔ We maximize V with Vmax

This algorithm must be applied to a problem that defines velocities addition multiplication etc etc

We can use PSO for computing weights for neural networks

initialize all positions randomly-> comptue the current fitness for every particle-> check if the current fitness is better than the local best for each partic