Swarm Intelligence
Particle Swarm Optimization

Christian Camacho Villalón
christian.camacho.villalon@ulb.ac.be
Federico Pagnozzi
federico.pagnozzi@ulb.ac.be

IRIDIA – Université Libre de Bruxelles (ULB)
Bruxelles, Belgium
Particle Swarm Optimisation

- Perturbative algorithm
  - A set of solutions are input to the algorithm from the beginning

- Originally proposed for continuous problems
  - Adaptations have been propose to tackle discrete and mixed variable problems

- Particles communicate among them and share information about the solutions they have found.
  - Topologies and models of influence.
Particle Swarm Optimisation

- **Solutions**: particle moving through the search space
  
  - Position \( x[] = (x_1, x_2, ..., x_n) \)
  
  - Velocity \( v[] = (v_1, v_2, ..., v_n) \)
  
  - Personal best position \( p[] = (p_1, p_2, ..., p_n) \)
  
  - Global best position \( g[] = (g_1, g_2, ..., g_n) \)

\[
\begin{align*}
  v_i(t + 1) &= v_i(t) + \psi_1 \ast U_1(p_i - x_i(t)) + \psi_2 \ast U_2(g_i - x_i(t)) \\
  x_i(t + 1) &= x_i(t) + v_i(t + 1)
\end{align*}
\]
Particle Swarm Optimisation

\[ v_i(t+1) = \omega \ast v_i(t) + \psi_1 \ast U_1(p_i - x_i(t)) + \psi_2 \ast U_2(g_i - x_i(t)) \]

\[ x_i(t+1) = x_i(t) + v_i(t+1) \]
Particle Swarm Optimisation

1. Initialize particles
2. While (!termination)
   3. Update global best  #Topology dependent
   4. Update velocity
   5. Update current position
   6. Update personal best
   7. End while
8. Return best solution
Particle Swarm Optimisation

\[ F_{\text{Rastrigin}} = \sum_{i=1}^{m} (x_i^2 - 10 \cos(2\pi x_i) + 10) \]

http://cg.kw.ac.kr/kang/pso/
Particle Swarm Optimisation

\[ F_{\text{Rastrigin}} = \sum_{i=1}^{m} (x_i^2 - 10 \cos(2\pi x_i) + 10) \]

http://cg.kw.ac.kr/kang/pso/
Particle Swarm Optimisation: Topologies

• Define a neighbourhood for the particles.
  - Gbest: all particles are neighbours.
  - Ring: Each particle has $n$ other neighbours.
Particle Swarm Optimisation: Topologies

- **Wheel**: one central particle.

- **Von Neumann**: Network of 2 dimensions, each particle connected up and down, left and right.
Particle Swarm Optimisation: Models of influence

- The model of influence determines which particle(s) will contribute to update the velocity vector of $p_i$ (aka its informants)

- **Best of neighbourhood**: only gbest or lbest

$$v_i(t+1) = \omega \cdot v_i(t) + \psi_1 \cdot U_1(p_i - x_i(t)) + \psi_2 \cdot U_2(g_i - x_i(t))$$
Particle Swarm Optimisation: Models of influence

- The model of influence determines which particle(s) will contribute to update the velocity vector of \( p_i \) (aka its informants)

- **Best of neighbourhood**: only \( g \text{best} \) or \( l \text{best} \)

- **Fully informed**: all neighbours

\[
v_i(t+1) = \chi [v_i(t) + \sum_{j=1}^{N} \psi_j \ast U_j(p_j - x_i(t))]
\]

\[
\psi_j = \frac{\psi}{|N_i|} \quad \chi = 0.7298
\]
Particle Swarm Optimisation: Inertia

\[ v_i(t+1) = \omega \cdot v_i(t) + \psi_1 \cdot U_1(p_i - x_i(t)) + \psi_2 \cdot U_2(g_i - x_i(t)) \]

- Controlling the balance of the search
  - Small inertia → more exploitative
  - Large inertia → more exploratory
Particle Swarm Optimisation: Inertia

• The value of inertia can vary during the search:
  - Example: Privilege exploration in initial iterations and exploitation at the end.

\[ v_i(t+1) = \omega(t) \cdot v_i(t) + \psi_1 \cdot U_1(p_i - x_i(t)) + \psi_2 \cdot U_2(g_i - x_i(t)) \]

  - Linear decreasing:

\[ \omega(t) = (\omega(0) - \omega(T)) \cdot \frac{T-t}{T} + \omega(T) \]

  - No linear decreasing:

\[ \omega(t+1) = \omega(T) - (\omega(T) - \omega(0)) \left( \frac{T-t}{T} \right)^\alpha \]