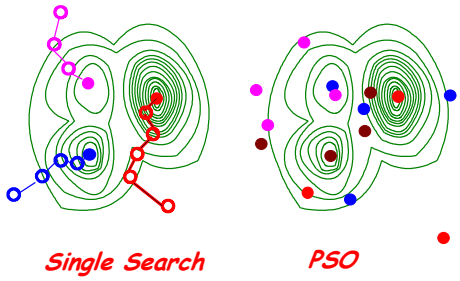
Particle Swarm Optimization

=

Coordination with Direct Communication

PSO vs SST



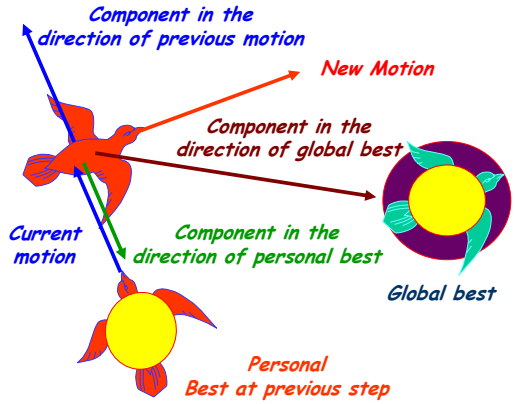
Single Search PSO

Particle Swarm Optimization

- *Inventors:* James Kennedy and Russell Eberhart
- An Algorithm originally developed to imitate the motion of a Flock of Birds, or insects
- Assumes Information Exchange (Social Interactions) among the search agents
- Basic Idea: Keep track of
  - Global Best
  - Self Best

How does it work?

- Problem:
  Find $\boldsymbol{X}$ which minimizes f($\boldsymbol{X}$)
- Particle Swarm:
  - ***Start:*** Random set of solution vectors
  - ***Experiment:*** Include randomness in the choice of new states.
  - ***Remember:*** Encode the information about good solutions.
  - ***Improvise:*** Use the 'experience' information to initiate search in a new regions



PSO Modeling

- Each solution vector is modeled as
  - The coordinates of a bird or a 'particle' in a 'swarm' flying through the search space
  - All the particles have a non-zero velocity and thus never stop flying and are always sampling new regions.
- Each 'particle' remembers
  - Where the global best and where the local best are.

- The search is guided by
  - The collective consciousness of the swarm
  - Introducing randomness into the dynamics in a controlled manner
- Particle Swarm Dynamics

$$\vec{x}(k+1) = \vec{x}(k) + \vec{v}(k)$$

$$\vec{v}(k+1) = w.\vec{v}(k) + r(0,a_1).(\vec{x}_{SelfBest}(k) - \vec{x}(k)) + r(0,a_2).(\vec{x}_{GroupBest}(k) - \vec{x}(k))$$

Particle Swarm Dynamics

$$\vec{x}(k+1) = \vec{x}(k) + \vec{v}(k)$$

non-zero velocity
PS never stop flying

$$\vec{v}(k+1) = w.\vec{v}(k) + r(0,a_1).(\vec{x}_{SelfBest}(k) - \vec{x}(k)) + r(0,a_2).(\vec{x}_{GroupBest}(k) - \vec{x}(k))$$

Inertia
Self consciousness of the swarm
Controlled randomness
The collective consciousness of the swarm

PSO

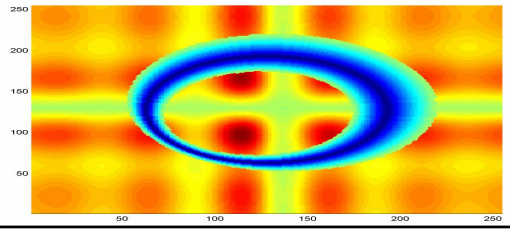
- where,
  - x is a solution vector 'particle' and v is the velocity of this particle
  - a_1 and a_2 are two scalars,
  - w is the inertia
  - $r(0,1)$ is a uniform random number generator between 0 and 1

Design Parameters

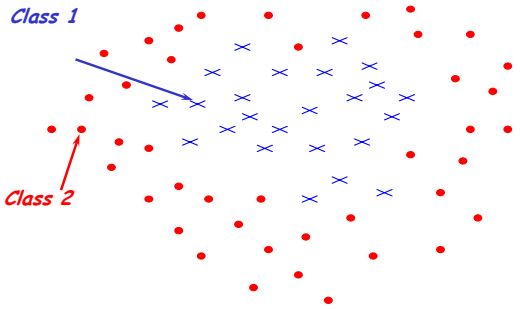
- a_1 and a_2
- w: Should be between [0.9 and 1.2]
  - High values of w gives a global search
  - Low values of w gives a local search
- v_{max}: To be designed according to the nature of the search surface.

Example: Boundary Identification (Edge detector)

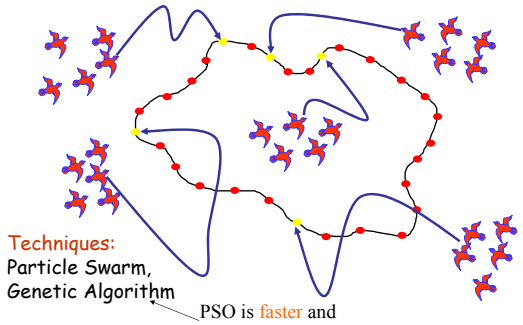
- To identify a subset of the search space(the boundary) with specific value
- Each flock finds one point on that boundary (edge)
- Flocks search sequentially



Border (Edge) Identification



Border (Edge) Identification

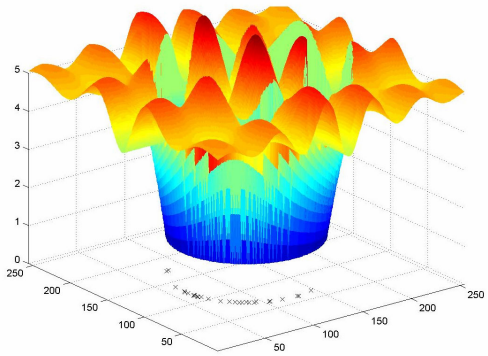


The Art of Fitness Function

- *To find points anywhere on the boundary*

Metric: |f(x)-boundary value|

Results - Case 1



The Art of Fitness Function

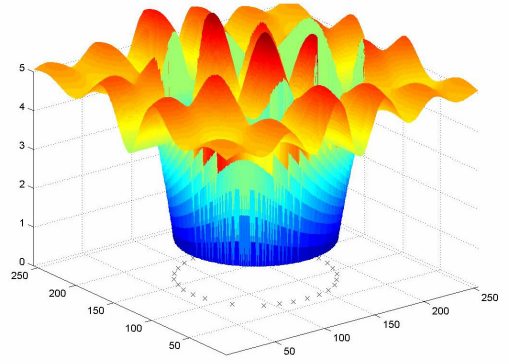
- *Distribute points uniformly on the boundary*

Metric:

|f(x)-boundary value| - Distance to closest neighbor

(to penalize proximity to neighbors)

Results - Case 2



The Art of Fitness Function

- Distribute points uniformly on the boundary close to current state

Metric:

|f(x)-boundary value| -Distance to closest neighbor + Distance to current state

(penalize proximity to neighbors, penalize distance from current state)

Results - Case 3