Particle Swarm Optimization

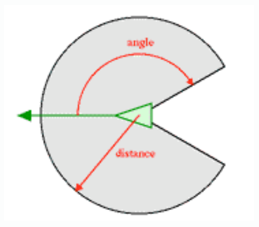
http://swarmintelligence.org/

Compare and integrate with existing heuristics

Motivation: Flocking

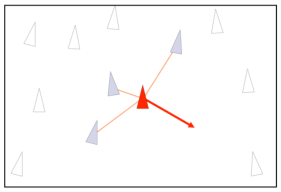
- Boids
  - 1986, Craig Reynolds
  - http://www.red3d.com/cwr/boids/
- Basic Flocking Model: 3 rules
  - Separation
  - Alignment
  - Cohesion

Agent Model



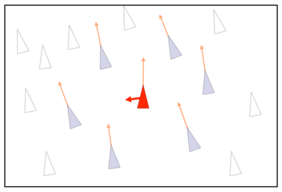
Collision Avoidance

- Rule 1: Avoid Collision with neighboring birds



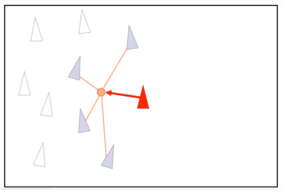
Velocity Matching

- Rule 2: Match the velocity of neighboring birds



Flock Centering

- Rule 3: Stay near neighboring birds

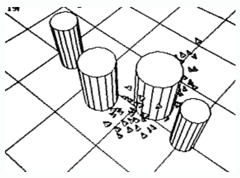


Flocking

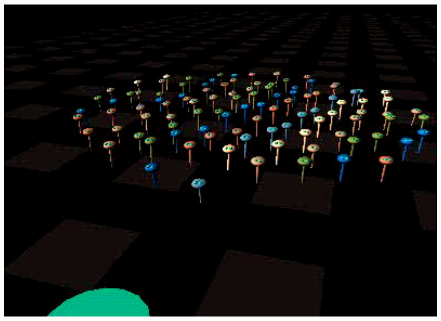
- Applications
  - Animated short (SIGGRAPH ’87)
    - Stanley and Stella in: Breaking the Ice
  - Movies
    - 1992 Tim Burton film **Batman Returns** – computer simulated bat swarms and penguin flocks
    - **Lion King** – the stampede
    - Object avoidance (many examples)

Obstacle Avoidance

- Models have predictive obstacle avoidance and goal seeking.
- Obstacle avoidance allowed boids to fly through environments while dodging static objects.
- Applications in computer animation, a low priority goal seeking behavior caused the flock to follow a scripted path.



Object Avoidance



Using the Metaphor

- Flock members are particles
- Environment is n-dimensional space
- Particles move through space, following a leader
- Influenced by "neighbors"
- Leads to …
  - Particle Swarm Optimization (PSO)

Particle Swarm Optimization

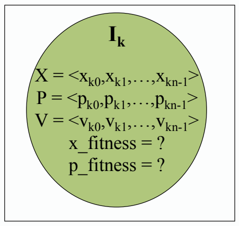
- PSO is a robust stochastic optimization technique based on the **movement** and **intelligence** of swarms.
- PSO applies the concept of **social interaction** to problem solving.
- It was developed in 1995 by James Kennedy (social-psychologist) and Russell Eberhart (electrical engineer).
- It uses a number of agents (particles) that constitute a swarm moving around in the search space looking for the best solution.
- Each particle is treated as a point in a N-dimensional space which adjusts its "flying" according to its own flying experience as well as the flying experience of other particles.

Particle Swarm Optimization

- Each particle keeps track of its coordinates in the solution space which are associated with the best solution (fitness) that has achieved so far by that particle. This value is called personal best, ***pbest***.
- Another best value that is tracked by the PSO is the best value obtained so far by any particle in the *neighborhood* of that particle. This value is called ***gbest***.
- The basic concept of PSO lies in accelerating each particle toward its pbest and the gbest locations, with a random weighted acceleration at each time step.

Anatomy of a Particle

- A particle is composed of:
  - **x-vector** records the current position (location) of the particle in the search space,
  - **p-vector** records the location of the best solution found so far by the particle, and
  - **v-vector** contains a gradient (direction) for which particle will travel in if undisturbed.
  - **x-fitness** records the fitness of the x-vector, and
  - **p-fitness** records the fitness of the p-vector.


I_k
$X = <x_{k0},x_{k1},\ldots,x_{kn-1}>$
$P = <p_{k0},p_{k1},\ldots,p_{kn-1}>$
$V = <v_{k0},v_{k1},\ldots,v_{kn-1}>$
x_fitness = ?
p_fitness = ?

PSO Search

- Particles live forever (unlike GA population members)
- So the question now is, “How does a particle move from on location in the search space to another?”
- This is done by simply adding the v-vector to the x-vector to get another x-vector ($X_i = X_i + V_i$).
- Once the particle computes the new X_i it then evaluates its new location. If x-fitness is better than p-fitness, then $P_i = X_i$ and p-fitness = x-fitness.

Particle Movement

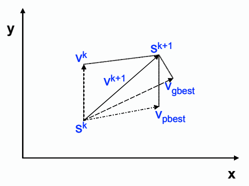
- Actually, we must adjust the v-vector before adding it to the x-vector as follows:

$$v_{id} = \omega * v_{id} + \varphi_1 * rnd() * (p_{id} - x_{id}) + \varphi_2 * rnd() * (p_{gd} - x_{id}) \quad (1)$$

$$x_{id} = x_{id} + v_{id} \quad (2)$$

- Where **i** is the particle,
- φ_1, φ_2 are learning rates governing the **cognition** and **social** components
- **ω** is an inertial factor
- Where **g** represents the index of the particle with the best p-fitness, and
- Where **d** is the dth dimension.

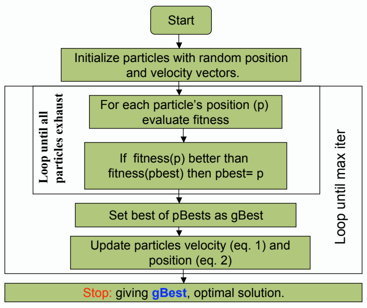
Particle Updating …



Concept of modification of a searching point by PSO

s^k : current searching point.
s^{k+1}: modified searching point.
v^k: current velocity.
v^{k+1}: modified velocity.
v_{pbest} : velocity based on pbest.
v_{gbest} : velocity based on gbest

Flow chart for PSO Algorithm


Start
Initialize particles with random position and velocity vectors.
For each particle's position (p) evaluate fitness
If fitness(p) better than fitness(pbest) then pbest= p
Loop until all particles exhaust
Set best of pBests as gBest
Update particles velocity (eq. 1) and position (eq. 2)
Loop until max iter
Stop: giving gBest, optimal solution.

Algorithm I

- System initialized with a population of random potential solutions.
- Each potential solution is assigned a randomized **'velocity'** and is called a particle. (It has position in the space, i.e. it is a point in the solution space and it has velocity. So it is analogous to a particle in physics which flies around in 3-space!)
- These particles are then **'flown'** through the (hyper) space of potential solutions.
- Each particle keeps track of the coordinates in the hyperspace for which it has achieved the best solution so far. (In the case of a NN that would be a particular set of weights.) and its best fitness (call it pBest) so far.

Algorithm II

- In the 'global' version of the Optimizer, is the overall best value, **gBest**, and its location (i.e. in the NN case, its set of weights). This particle is the leader.
- At each time step the 'velocity' of each particle is changed (accelerated) towards its **pBest** and **gBest** fellows. This acceleration is weighted by a random term. (The idea is that all the particles swarm towards where the current best solutions are. The random factor prevents the swarm getting stuck in the wrong place -- insects around a light.)
- A new position in the solution space is calculated for each particle by adding the new velocity value to each component of the particle's position vector. (In the NN case each weight of each potential solution would be adjusted).
- The user specifies an acceleration constant and a maximum velocity.

Swarm Types

- In: [Kennedy, J. (1997), "The Particle Swarm: Social Adaptation of Knowledge", Proceedings of the 1997 International Conference on Evolutionary Computation, pp. 303-308, IEEE Press.]
- Kennedy identifies 4 types of PSO based on φ_1 and φ_2 .
- Given: $v_{id} = \omega * v_{id} + \varphi_1 * rnd() * (p_{id} - x_{id}) + \varphi_2 * rnd() * (p_{gd} - x_{id})$

  $x_{id} = x_{id} + v_{id}$

  - Full Model ($\varphi_1, \varphi_2 > 0$)
  - Cognition Only ($\varphi_1 > 0$ and $\varphi_2 = 0$),
  - Social Only ($\varphi_1 = 0$ and $\varphi_2 > 0$)
  - Selfless ($\varphi_1 = 0$, $\varphi_2 > 0$, and $g \neq i$)

Vector Form of Equations …

$$\vec{x}_i(t) = f(\vec{x}_i(t-1), \vec{v}_i(t-1), \vec{p}_i, \vec{p}_g)$$

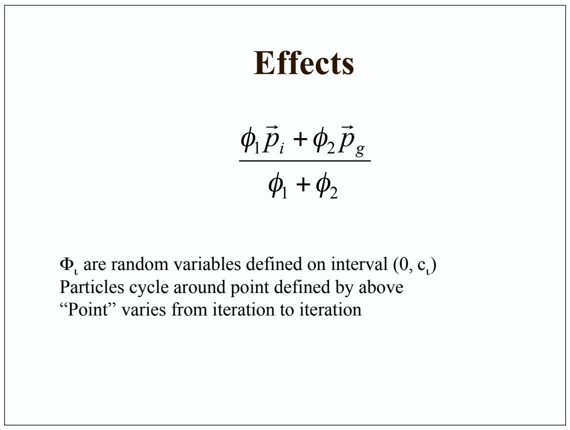
Particle position a function of particle and swarm best solutions.

$$\begin{Bmatrix} \vec{v}_i(t) = \omega \vec{v}_i(t-1) + \phi_1(\vec{p}_i - \vec{x}_i(t-1)) + \phi_2(\vec{p}_g - \vec{x}_i(t-1)) \\ \vec{x}_i(t) = \vec{x}_i(t-1) + \vec{v}_i(t) \end{Bmatrix}$$

if $v_{id} > V_{max}$ then $v_{id} = V_{max}$

if $v_{id} < -V_{max}$ then $v_{id} = -V_{max}$

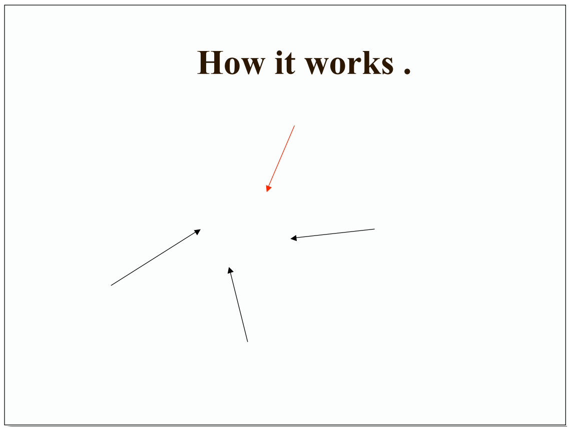
Velocity Constraints



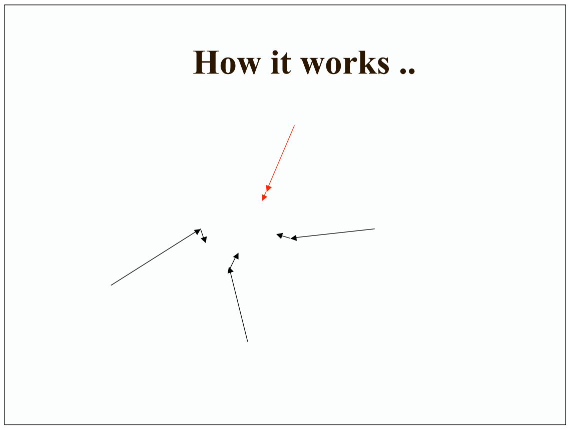
Effects

$$\frac{\phi_1 \vec{p}_i + \phi_2 \vec{p}_g}{\phi_1 + \phi_2}$$

Φ_i are random variables defined on interval (0, c_i)
Particles cycle around point defined by above
“Point” varies from iteration to iteration

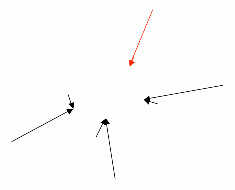


How it works .

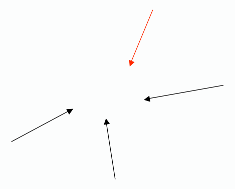


How it works ..

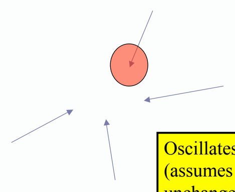
How it works ….



How it works ….



How it works …..



Oscillates around best point (assumes that this remains unchanged here)

Algorithm Issues

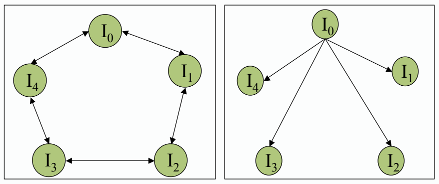
- Several related issues:
  - Controlling velocities (determining the best value for Vmax),
  - Swarm Size,
  - Neighborhood Size,
  - Updating X and Velocity Vectors,
  - Robust Settings for φ_1 and φ_2
- An Off-The-Shelf PSO:
  - Carlisle, A. and Dozier, G. (2001). "An Off-The-Shelf PSO", *Proceedings of the2001 Workshop on Particle Swarm Optimization*, pp. 1-6, Indianapolis, IN. (http://antho.huntingdon.edu/publications/Off-The-Shelf_PSO.pdf)

Initial Velocity Considerations

- Initially, the values of the velocity vectors are randomly generated with the range [-Vmax, Vmax] where Vmax is the maximum value that can be assigned to any v_{id}.

Swarm Topology

- Two basic topologies used in the literature
  - Ring Topology (neighborhood of 3)
  - Star Topology (global neighborhood)

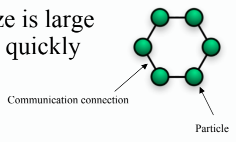


Neighbors …

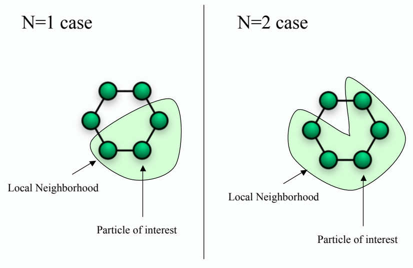
- Key element of the algorithm is how to define the neighborhoods
- Both the size and the topology of the neighborhoods is important, they define the social aspect of the algorithm
- Large neighborhoods result in localized searches
- Smaller neighborhoods result in greater search variety
- Topologies decide how the information is passed amongst the particles

Ring Topology

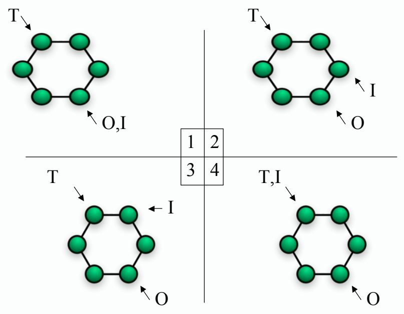
- Each particle has two nearest neighbors
- If the neighborhood size is small then it takes a long time for information to travel
- If neighborhood size is large information travels quickly



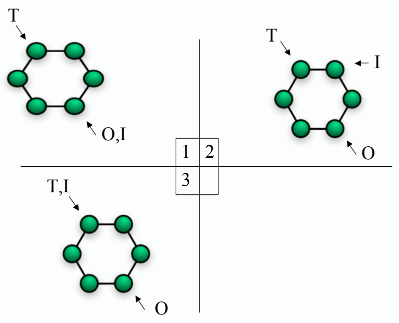
Ring Topology

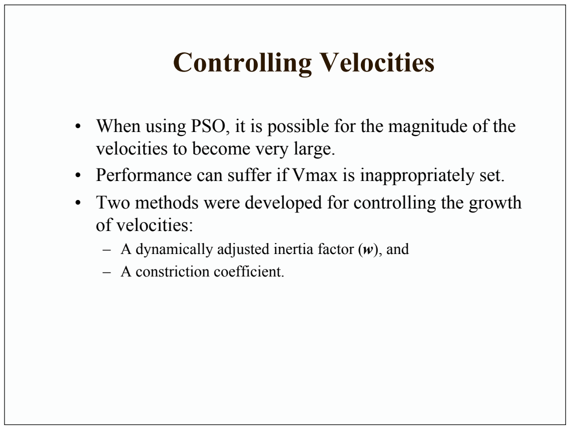


Information Passing (N=1)



Information Passing (N=2)





Controlling Velocities

- When using PSO, it is possible for the magnitude of the velocities to become very large.
- Performance can suffer if Vmax is inappropriately set.
- Two methods were developed for controlling the growth of velocities:
  - A dynamically adjusted inertia factor (***w***), and
  - A constriction coefficient.

Inertia Factor

- When inertia factor is used, equation for updating velocities becomes:

```
v_id = ω*v_id + φ_1*rnd()*(p_id-x_id)
              + φ_2*rnd()*(p_gd-x_id)
```

- Where ω is initialized to 1.0 and is gradually reduced over time (measured by cycles through the algorithm).
  - Analogous to quenching in SA

Functional Form of Inertia Factor

The following weighting function is usually utilized:

$\omega = \omega_{max} - [(\omega_{max} - \omega_{min}) * iter]/maxIter$

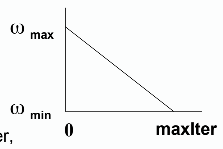
Where:

ω_{max} = initial weight,

ω_{min} = final weight,

maxIter = maximum iteration number,

iter = current iteration number.

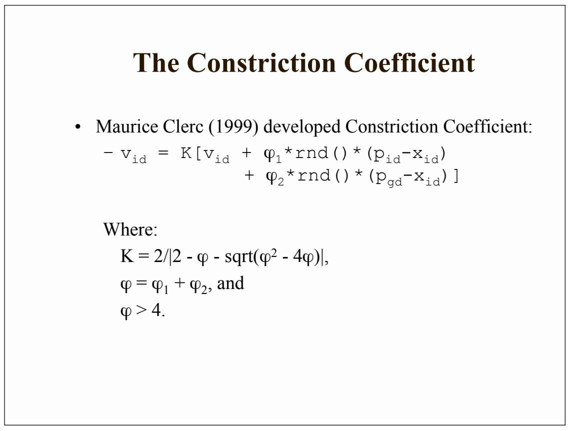


Inertia Weight Factor

A large inertia weight (ω) facilitates a global search while a small inertia weight facilitates a local search.

By linearly decreasing the inertia weight from a relatively large value to a small value through the course of the PSO run gives the best PSO performance compared with fixed inertia weight settings.

Larger ω → greater *global* search ability
Smaller ω → greater *local* search ability.



The Constriction Coefficient

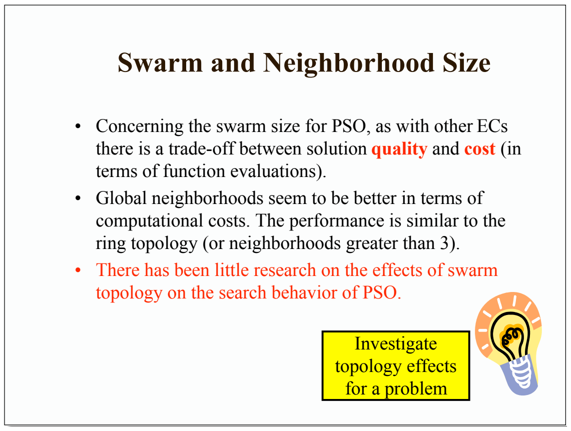
- Maurice Clerc (1999) developed Constriction Coefficient:
  - $v_{id} = K[v_{id} + \varphi_1 * rnd() * (p_{id} - x_{id}) + \varphi_2 * rnd() * (p_{gd} - x_{id})]$

  Where:

  $K = 2/|2 - \varphi - \text{sqrt}(\varphi^2 - 4\varphi)|$,

  $\varphi = \varphi_1 + \varphi_2$, and

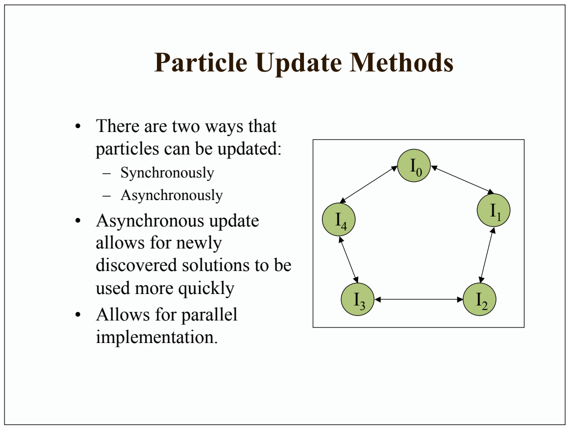
  $\varphi > 4$.



Swarm and Neighborhood Size

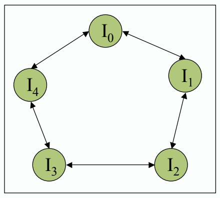
- Concerning the swarm size for PSO, as with other ECs there is a trade-off between solution **quality** and **cost** (in terms of function evaluations).
- Global neighborhoods seem to be better in terms of computational costs. The performance is similar to the ring topology (or neighborhoods greater than 3).
- There has been little research on the effects of swarm topology on the search behavior of PSO.

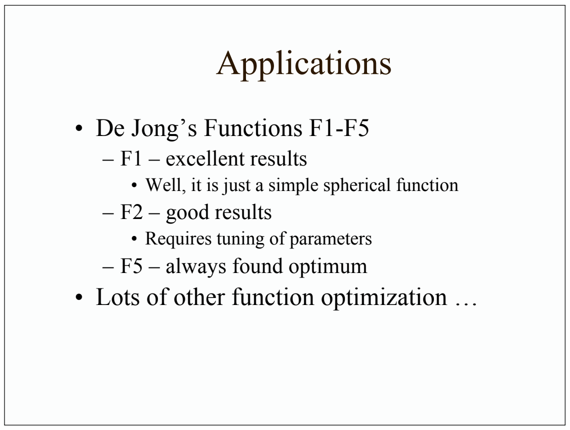
Investigate topology effects for a problem



Particle Update Methods

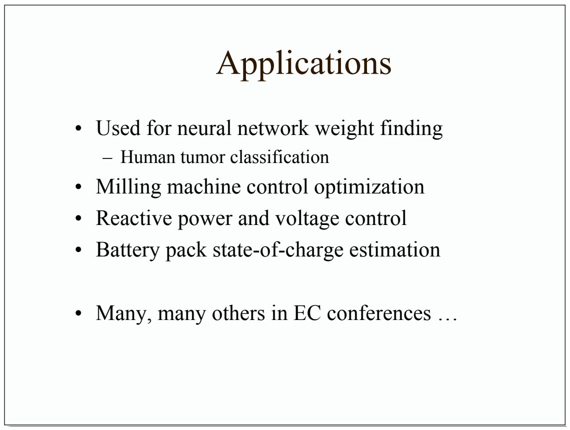
- There are two ways that particles can be updated:
  - Synchronously
  - Asynchronously
- Asynchronous update allows for newly discovered solutions to be used more quickly
- Allows for parallel implementation.





Applications

- De Jong’s Functions F1-F5
  - F1 – excellent results
    - Well, it is just a simple spherical function
  - F2 – good results
    - Requires tuning of parameters
  - F5 – always found optimum
- Lots of other function optimization …



Applications

- Used for neural network weight finding
  - Human tumor classification
- Milling machine control optimization
- Reactive power and voltage control
- Battery pack state-of-charge estimation

- Many, many others in EC conferences …