IM5110701 Metaheuristics

Dr. Hendri Sutrisno

Institute of Statistical Science

Academia Sinica

Dr. Chao-Lung Yang

Department of Industrial Management

National Taiwan University of Science and Technology

Unit 5: Particle Swarm Optimization

Sean Luke, Essentials of Metaheuristics, Second Edition

Numerical example and some slides are borrowed from Dr. Y.-D. Yang's course slides of Application of Computational Intelligence in Engineering

Video source: https://www.youtube.com/watch?v=V4f_1_r80RY

Swarm Intelligent From Nature

- Fish school. Birds flock. Bees swarm. A combination of real-time, biological systems blends knowledge, wisdom, opinions and intuition to unify intelligence.

Image source: https://www.flyingmag.com/how-to-become-swarm-technology-researcher/

Image source: https://www.cio.com/article/234859/why-swarm-intelligence-enhances-business-and-bitcoin.html

Swarm Intelligence

- Swarm of creatures could achieve things that no individual could
  - No central control
  - **Cooperation** and **self-organization**
  - Simple, and yet powerful rule for individual

Introduction (1)

- First described in 1995 by James Kennedy and Russell Eberhart
- Implement the underlying rules that enable large numbers of organisms (birds, fishes, herds) to move synchronously, often changing direction suddenly, scattering and regrouping
- A stochastic optimization technique somewhat similar to evolutionary algorithms but different in an important way.

J. Kennedy and R. Eberhart, "Particle swarm optimization," *Proceedings of ICNN'95 - International Conference on Neural Networks*, 1995, pp. 1942-1948 vol.4, doi: 10.1109/ICNN.1995.488968.

Introduction (2)

- Unlike other population-based methods, PSO does not resample populations to produce new ones: it has no selection of any kind.
- Instead, PSO maintains a single static population whose members are Tweaked in response to new discoveries about the space. The method is essentially a form of directed mutation.

PSO Theory (1)

- Each particle iteratively moves across the search space
- Attracted to the position (location) of the best fitness (evaluation of the objective function) historically achieved by the particle itself (local best; pBest) and by the best among the neighbors of the particle (global best; gBest).

PSO Theory (2)

- In essence, each particle continuously focuses and refocuses the effort of its search according to both local and global best.
- This behavior mimics the cultural adaptation of a biological agent in a swarm: it evaluates its own position based on certain fitness criteria, compares with others, and imitates the best in the entire swarm

Particle

- A particle consists of two parts:
  - The particle's location in space, $\vec{x} = \langle x_1, x_2, \ldots \rangle$. This is the equivalent, in evolutionary algorithms, of the individual's genotype.
  - The particle's velocity, $\vec{v} = \langle v_1, v_2, \ldots \rangle$. This is the speed and direction at which the particle is traveling each timestep. Put another way, if $\vec{x}^{(t-1)}$ and $\vec{x}^{(t)}$ are the locations in space of the particle at times $t-1$ and t respectively, then at time t, $\vec{v} = \vec{x}^{(t)} - \vec{x}^{(t-1)}$
- Each particle starts at a random location and with a random velocity vector, often computed by choosing two random points in the space and using half the vector from one to the other (other options are a small random vector or a zero vector).

Particle Best vs. Social Best

- pBest: The fittest known location $\vec{x}^*$ that $\vec{x}$ has discovered so far.
- gBest: The fittest known location $\vec{x}^!$ that has been discovered by anyone (entire of swarm) so far.

Basic PSO Flow (1)

- Initialize a population of particles in the **search space**
  - Provide each particle with a random location and a random velocity within the N-dimensional space
- Evaluate each particle's fitness, based on the **objective value**
- If the fitness is better than the particle's best experience (pBest), save the location vector for the particle as pBest

Basic PSO Flow (2)

- If the fitness is better than the best in the entire population (gBest), save the location vector for the particle as gBest
- Update the particle's velocity and location based on pBest and gBest

Parameters of PSO (1)

- This implementation of the algorithm relies on five parameters
  - α (inertia weight): how much of the original velocity is retained, usually between 0 and 1
  - β: how much of the personal best is mixed in. If β is large, particles tend to move more towards their own personal bests rather than towards global bests. This breaks the swarm into a lot of separate hill-climbers rather than a joint searcher.

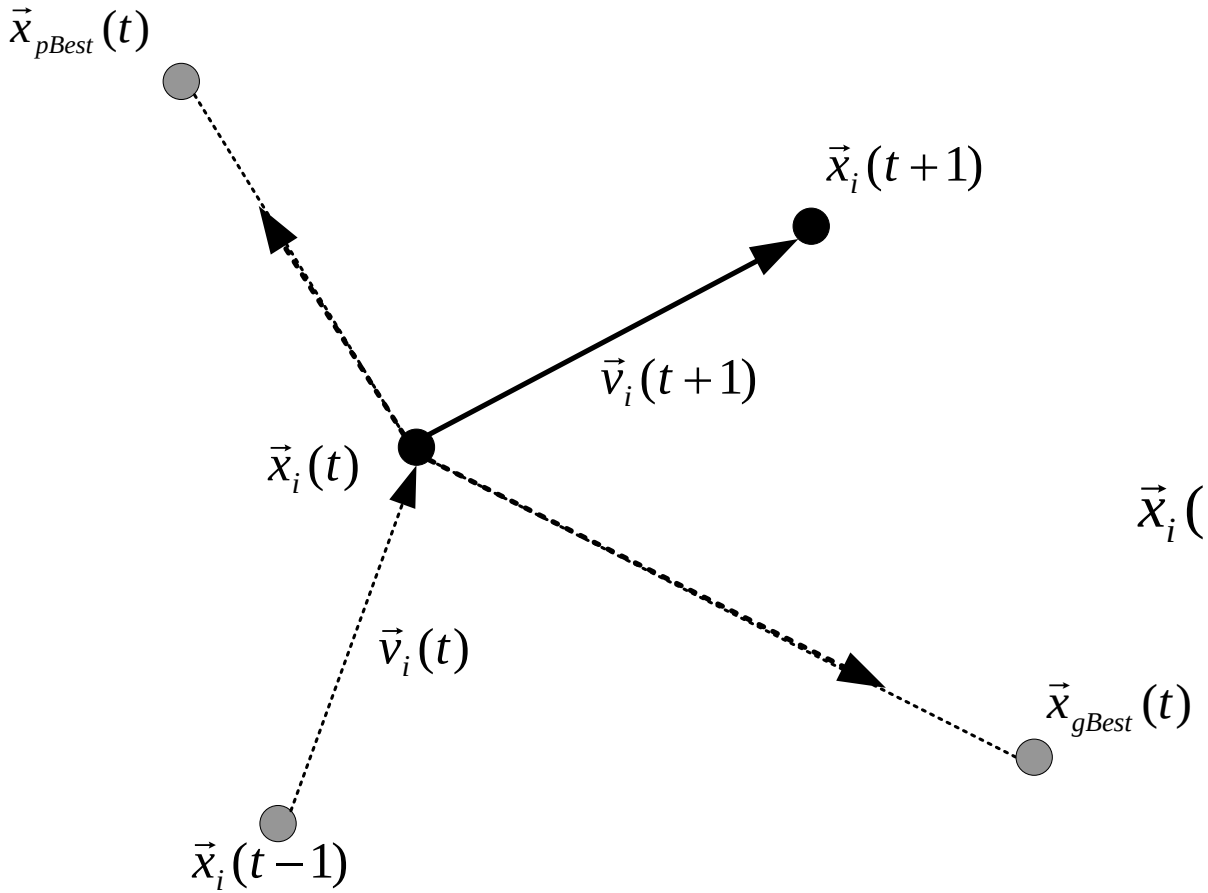
Parameters of PSO (2)

- δ: how much of the global best is mixed in. If δ is large, particles tend to move more towards the best known region. This converts the algorithm into one large hill-climber rather than separate hill-climbers. Perhaps because this threatens to make the system highly exploitative, δ is often set to 0 in modern implementations.
- ε: how fast the particle moves. If ε is large, the particles make big jumps towards the better areas—and can jump over them by accident. Thus a big ε allows the system to move quickly to best-known regions, but makes it hard to do fine-grained optimization. Just like in hill-climbing. Most commonly, ε is set to 1.

Update the Velocity

- $\vec{v}_i^{(t+1)} = \alpha \vec{v}_i^{(t)} + b(\vec{x}^* - \vec{x}^{(t)}) + d(\vec{x}^! - \vec{x}^{(t)})$
  - α is inertia weight
  - b is random number from 0 to β (randomness r × acceleration c)
  - d is random number from 0 to δ (randomness r × acceleration c)

Update the Location



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

Numerical Example

- Maximize $f(x,y) = x^{0.5} - 1/y$; $0 \leqq (x,y) \leqq 5$
- We will use this example to illustrate a PSO algorithm

Numerical Example (1)

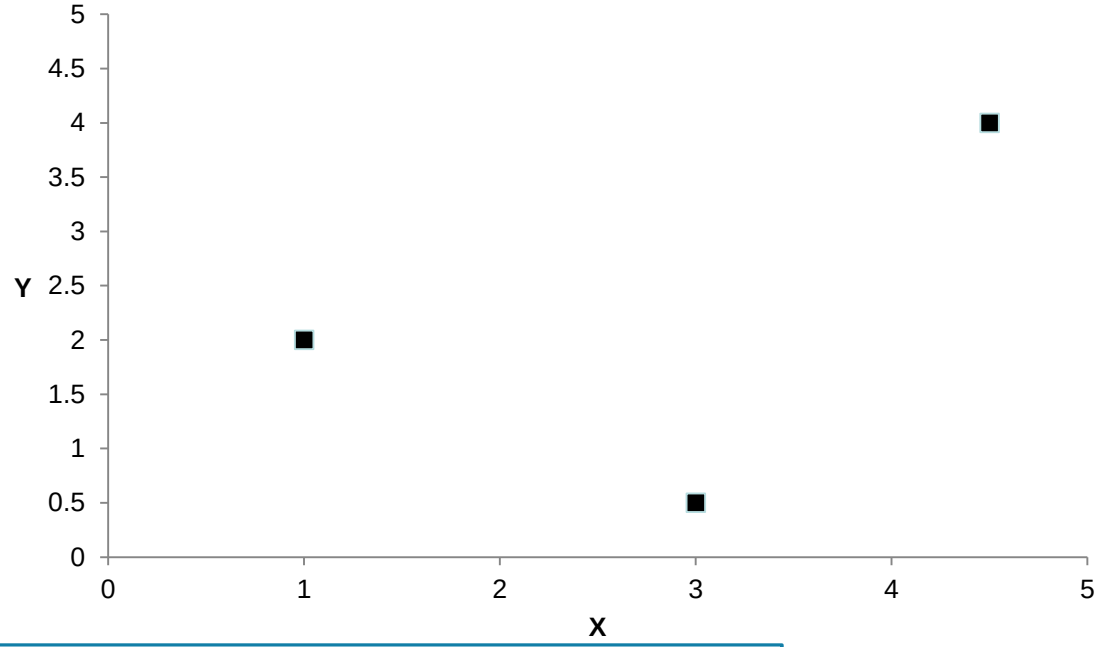
- Randomly initialize locations and velocities

Location

x	y
1	2
4.5	4
3	0.5

Velocity

x	y
0.83	-0.51
-0.67	0.94
0.45	0.89



Assume the initial velocity is randomly generated within range (-1,1)

Numerical Example (2)

Iteration 1: Calculate Fitness

Location		
x	y	Fitness
1	2	0.500
4.5	4	1.871
3	0.5	-0.268

Numerical Example (3)

Iteration 1: Update pBest and pBestLoc

This is the first iteration, so pBest will be the current fitness

pBest

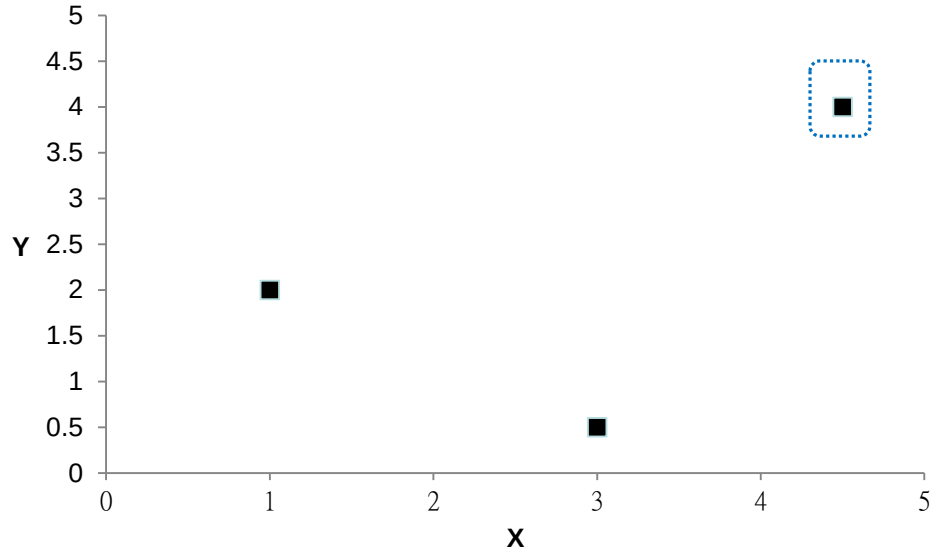
x	y	Fitness
1	2	0.500
4.5	4	1.871
3	0.5	-0.268

Numerical Example (4)

Iteration 1: Determine gBest and gBestLoc

gBest

x	y	Fitness
1	2	0.500
4.5	4	1.871
3	0.5	-0.268



Numerical Example (5)

Iteration 1: Calculate velocities

Velocity						
x	y		α	previous velocity	c_1	r_2 c_2
3.49	1.01	--->	1x(0.83,-0.51) + r_1x2x(0,0) +			0.38x2x[(4.5,4)-(1,2)]
-0.67	0.94	--->	global best; 1x(-0.67,0.94)			
1.86	4.18	--->	1x(0.45,0.89) + r_1x2x(0,0) +			0.47x2x[(4.5,4)-(3,0.5)]

Location

x	y
1	2
4.5	4
3	0.5

Velocity

x	y
0.83	-0.51
-0.67	0.94
0.45	0.89

pBest

x	y	Fitness
1	2	0.500
4.5	4	1.871
3	0.5	-0.268

gBest

x	y	Fitness
4.5	4	1.871

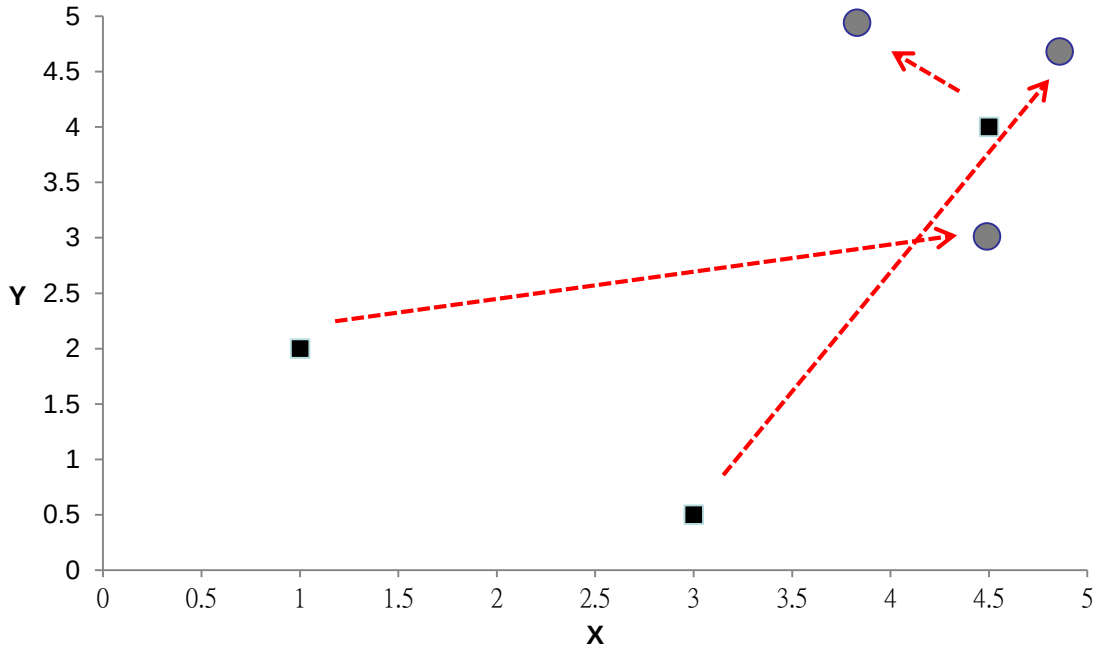
Numerical Example (6)

Iteration 1: Calculate locations

New Location			Location			Velocity	
x	y		x	y		x	y
4.49	3.01	=	1	2	+	3.49	1.01
3.83	4.94		4.5	4		-0.67	0.94
4.86	4.68		3	0.5		1.86	4.18

Numerical Example (7)

New Location			Location			Velocity	
x	y		x	y		x	y
4.49	3.01		1	2		3.49	1.01
3.83	4.94	=	4.5	4	+	-0.67	0.94
4.86	4.68		3	0.5		1.86	4.18



Numerical Example (8)

Iteration 2: Calculate fitness

Location		
x	y	Fitness
4.49	3.01	1.787
3.83	4.94	1.755
4.86	4.68	1.991

Numerical Example (9)

Iteration 2: Update pBest and pBestLoc

Original pBest

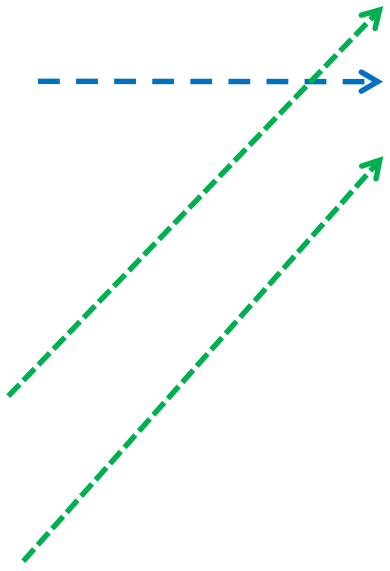
x	y	Fitness
1	2	0.500
4.5	4	1.871
3	0.5	-0.268

Location

x	y	Fitness
4.49	3.01	1.787
3.83	4.94	1.755
4.86	4.68	1.991

New pBest

x	y	Fitness
4.49	3.01	1.787
4.5	4	1.871
4.86	4.68	1.991

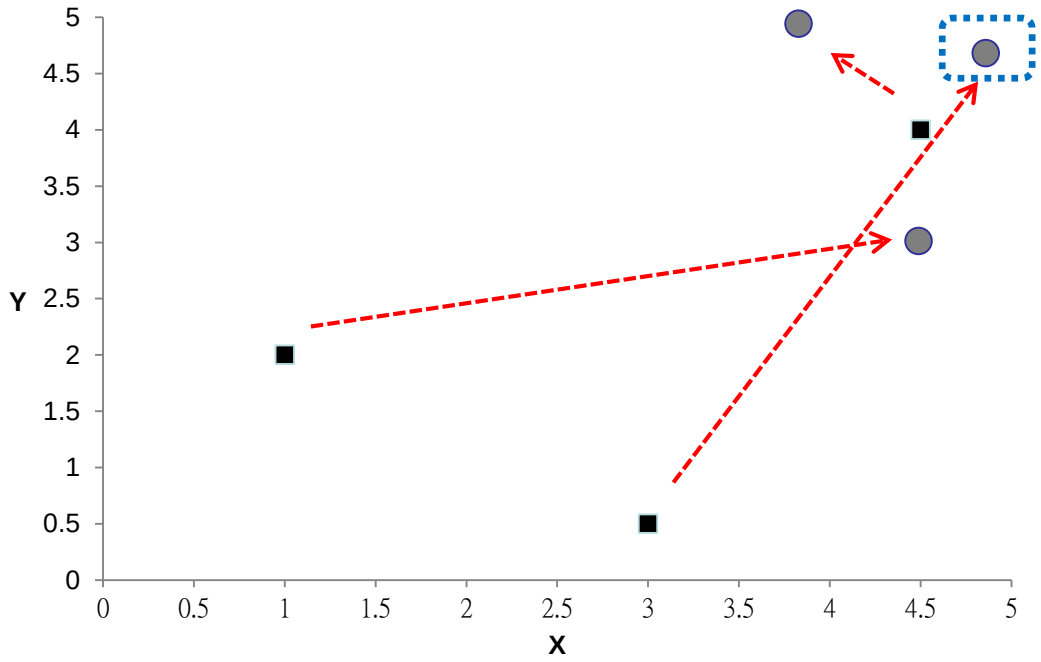


Numerical Example (10)

Iteration 2: Determine gBest and gBestLoc

gBest

x	y	Fitness
4.49	3.01	1.787
4.5	4	1.871
4.86	4.68	1.991



Constrained Velocity and Location

- If the velocity would lead the particle to move beyond boundaries, what should we do?
- First, the velocity may be limited
- Second, whether the location is allowed to move outside the boundary needs special caution

Damping Limit for Velocity

- To prevent undesired oscillations (i.e., the velocity may expand wider and wider until approaching infinity), the velocity is often dampened by an upper limit $v^{\max}$

$$v(i,t) = \frac{v(i,t)}{|v(i,t)|} v^{\max} \quad \text{if } |v(i,t)| > v^{\max}$$

$v^{\max}$ is smaller than the domain of the search space

Treatment for Boundary Violation

- Do not allow any particle to move outside, infeasible solutions
- Force particle to stick to the boundary; repair the location if necessary
- Let's continue iteration 2 of the numerical example to repair the location

Numerical Example (11)

Recap

Velocity

x	y
3.49	1.01
-0.67	0.94
1.86	4.18

Location

x	y	Fitness
4.49	3.01	1.787
3.83	4.94	1.755
4.86	4.68	1.991

pBest

x	y	Fitness
4.49	3.01	1.787
4.5	4	1.871
4.86	4.68	1.991

gBest

x	y	Fitness
4.86	4.68	1.991

Numerical Example (12)

Iteration 2: Calculate velocities

Velocity

$$\vec{v}_i(t+1) = w \times \vec{v}_i(t) + r_1 c_1 (\vec{x}_{pBest} - \vec{x}_i(t)) + r_2 c_2 (\vec{x}_{gBest} - \vec{x}_i(t))$$

x	y	
4.038	3.482	---> 1x(3.49,1.01) + 0.17x2x[(4.49,3.01)-(4.49,3.01)] + 0.74x2x[(4.86,4.68)-(4.49,3.01)]
0.151	0.286	---> 1x(-0.67,0.94) + 0.29x2x[(4.5,4)-(3.83,4.94)] + 0.21x2x[(4.86,4.68)-(3.83,4.94)]
1.860	4.180	---> 1x(1.86,4.18) + 0.68x2x[(4.86,4.68)-(4.86,4.68)] + 0.11x2x[(4.86,4.68)-(4.86,4.68)]

Numerical Example (13)

Iteration 2: Velocity damping; e.g., $v^{max} = 2$

Velocity		Damped Velocity	
x	y	x	y
4.038	3.482	2	2
0.151	0.286	0.151	0.286
1.860	4.180	1.860	2

Numerical Example (14)

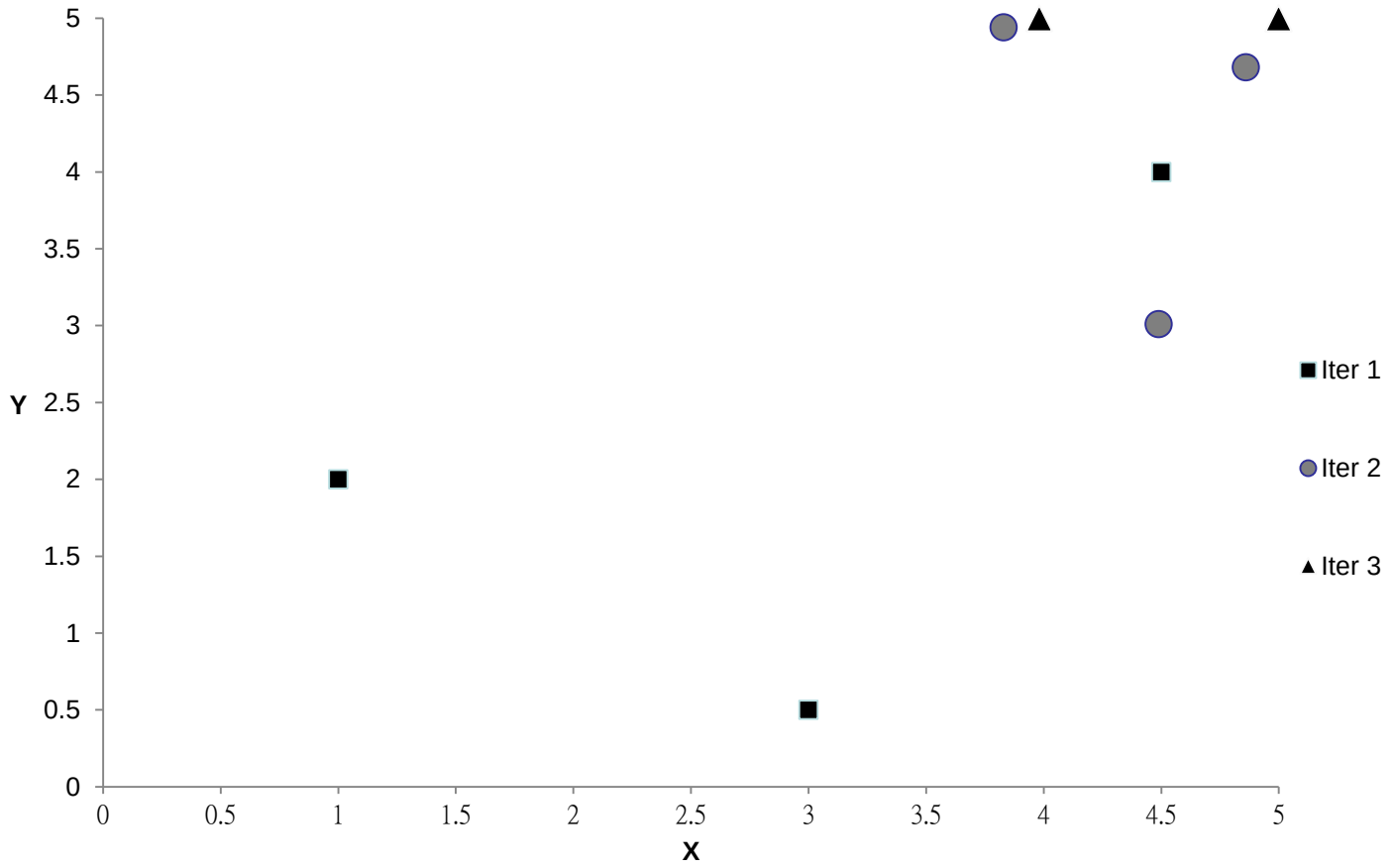
Iteration 2: Repair locations, e.g., ≦5

New Location			Location			Velocity	
x	y		x	y		x	y
6.490	5.01	=	4.49	3.01	+	2	2
3.981	5.226		3.83	4.94		0.151	0.286
6.720	6.680		4.86	4.68		1.860	2

Repaired Location

x	y
5.000	5.000
3.981	5.000
5.000	5.000

Numerical Example (15): End of Iteration 2



Still need velocity limit?

- If we will constrain the location after all, why do we need to set the damping limit for velocity?
- Because it will influence the velocity in the subsequent iterations

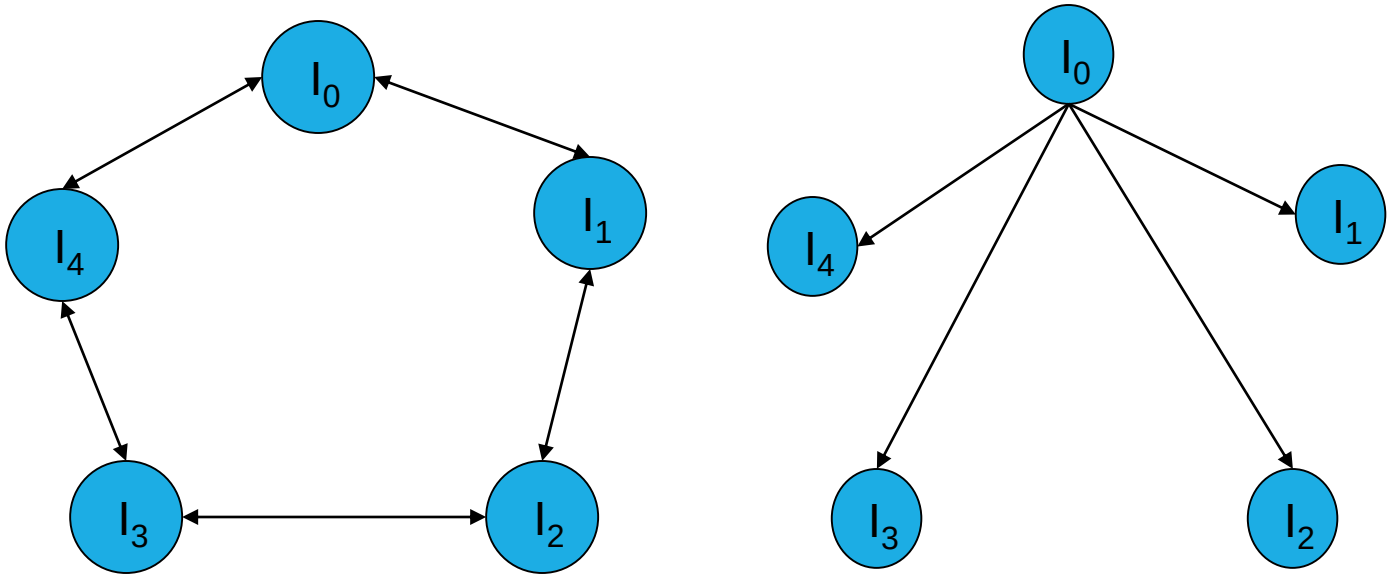
Neighborhoods and Topologies

- The topology of the swarm defines the subset of particles with which each particle can exchange information.
- This topology allows all particles to communicate with all the other particles, thus the whole swarm share the same best position g from a single particle.

Kennedy, J.; Mendes, R. (2002). *Population structure and particle swarm performance*. *Evolutionary Computation, 2002. CEC'02. Proceedings of the 2002 Congress on*. Vol. 2. pp. 1671–1676 vol.2.

Swarm Topology

- In PSO, there have been two basic topologies used in the literature
  - Ring Topology (neighborhood of 3)
  - Star Topology (global neighborhood)

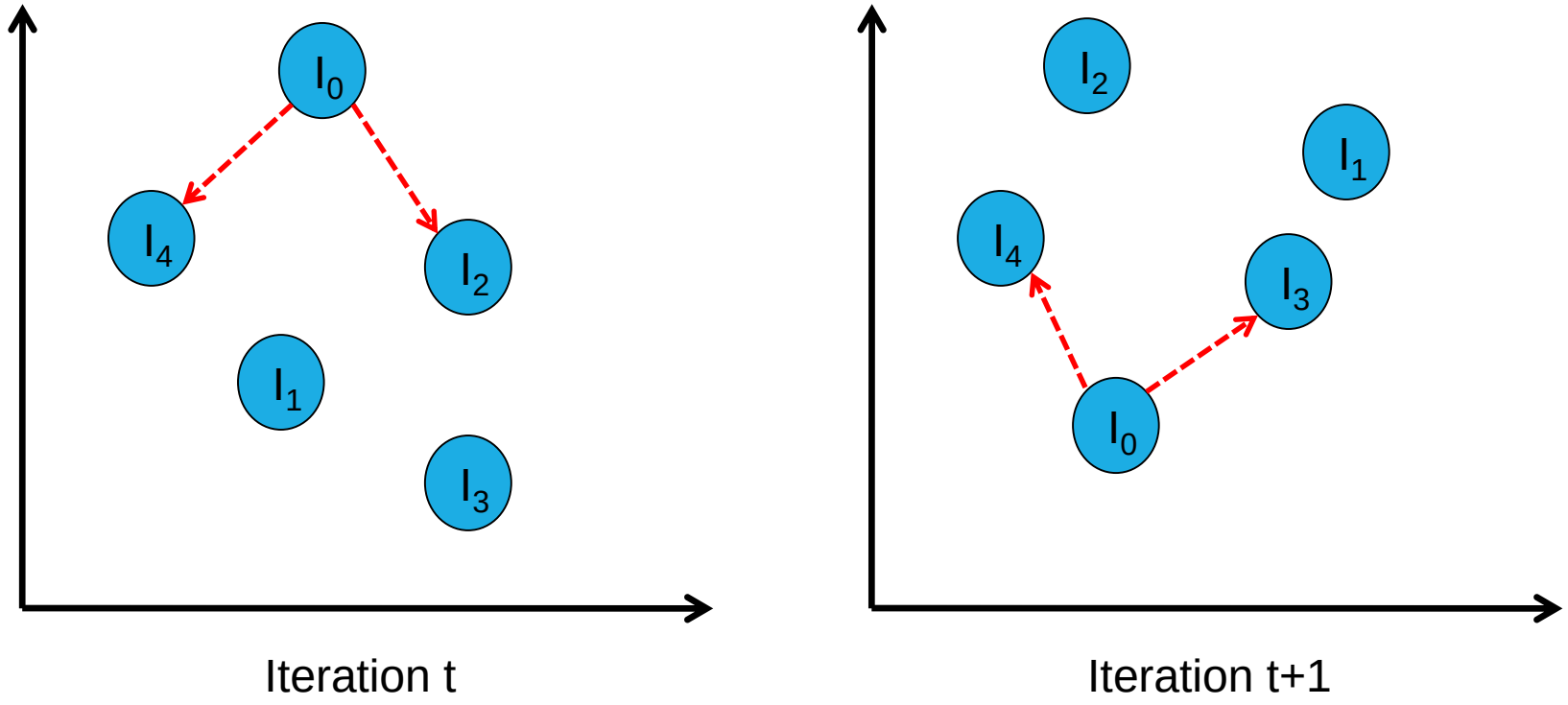


Particle Neighborhood (1)

- The neighborhood of a particles can be determined by
  - Pre-specified ID
  - Relative geographic positions in the search space
  - Ranks of particles in terms of fitness values

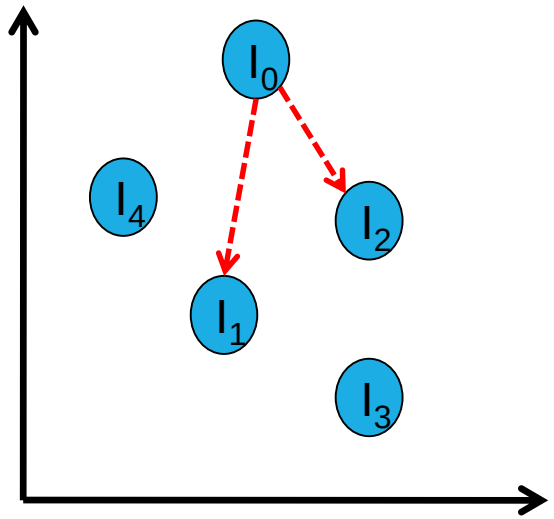
Particle Neighborhood (2)

- Relative geographic positions



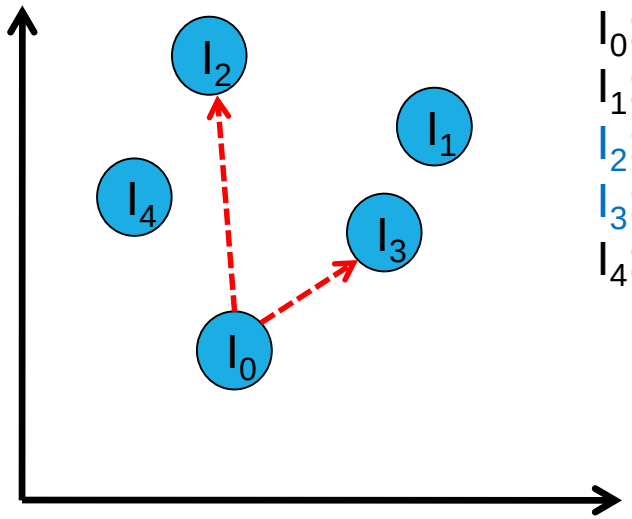
Particle Neighborhood (3)

- Ranks of particles in terms of fitness values



Iteration t

Fitness:
I_0: 16
I_1: 20
I_2: 15
I_3: 12
I_4: 10



Iteration t+1

Fitness:
I_0: 17
I_1: 13
I_2: 16
I_3: 22
I_4: 12

Informants for Sharing

- However, this approach might lead the swarm to be trapped into a local minimum, thus different topologies have been used to control the flow of information among particles.
- For instance, in local topologies, particles only share information with a subset of particles (informants).
  - This subset can be a geometrical one – for example "the m nearest particles" – or, more often, a social one, i.e. a set of particles that is not depending on any distance.
- In such cases, the PSO variant is said to be local best (vs global best for the basic PSO).

With Informant

- The fittest known location $\vec{x}^+$ that any of the informants of $\vec{x}$ have discovered so far. In early versions of the algorithm, particles were assigned “grid neighbors” which would inform them about known best-so-far locations. Nowadays the informants of $\vec{x}$ are commonly a small set of particles chosen randomly each iteration. $\vec{x}$ is always one of its own informants.

Parameters of PSO with Informants (1)

- This implementation of the algorithm relies on five parameters
  - α (inertia weight): how much of the original velocity is retained.
  - β: how much of the personal best is mixed in. If β is large, particles tend to move more towards their own personal bests rather than towards global bests. This breaks the swarm into a lot of separate hill-climbers rather than a joint searcher.
  - γ: how much of the informants' best is mixed in. The effect here may be a mid-ground between β and γ. The number of informants is also a factor (assuming they're picked at random): more informants is more like the global best and less like the particle's local best.

Parameters of PSO with Informants (2)

- δ: how much of the global best is mixed in. If d is large, particles tend to move more towards the best known region. This converts the algorithm into one large hill-climber rather than separate hill-climbers. Perhaps because this threatens to make the system highly exploitative, δ is often set to 0 in modern implementations.
- ε: how fast the particle moves. If ε is large, the particles make big jumps towards the better areas—and can jump over them by accident. Thus a big ε allows the system to move quickly to best-known regions, but makes it hard to do fine-grained optimization. Just like in hill-climbing. Most commonly, ε is set to 1.

Update the Velocity with Informants

- $\vec{v}_i^{(t+1)} = \alpha\vec{v}_i^{(t)} + b(\vec{x}^{*} - \vec{x}^{(t)}) + c(\vec{x}^{+} - \vec{x}^{(t)}) + d(\vec{x}^{!} - \vec{x}^{(t)})$
  - α is inertia weight
  - b is random number from 0 to β (randomness r × acceleration)
  - c is random number from 0 to γ (randomness r × acceleration)
  - d is random number from 0 to δ (randomness r × acceleration)

- This Pseudocode considers the more general version of PSO with informants

Algorithm 39 *Particle Swarm Optimization (PSO)*

```
swarmsize ← desired swarm size
α ← proportion of velocity to be retained
β ← proportion of personal best to be retained
γ ← proportion of the informants' best to be retained
δ ← proportion of global best to be retained
ϵ ← jump size of a particle

P ← {}
for swarmsize times do
    P ← P ∪ {new random particle x⃗ with a random initial velocity v⃗}
Best⃗ ← □
repeat
    for each particle x⃗ ∈ P with velocity v⃗ do
        AssessFitness(x⃗)
        if Best⃗ = □ or Fitness(x⃗) > Fitness(Best⃗) then
            Best⃗ ← x⃗
    for each particle x⃗ ∈ P with velocity v⃗ do          ▷ Determine how to Mutate
        x⃗* ← previous fittest location of x⃗
        x⃗+ ← previous fittest location of informants of x⃗   ▷ (including x⃗ itself)
        x⃗! ← previous fittest location any particle
        for each dimension i do
            b ← random number from 0.0 to β inclusive
            c ← random number from 0.0 to γ inclusive
            d ← random number from 0.0 to δ inclusive
            v_i ← αv_i + b(x*_i − x_i) + c(x+_i − x_i) + d(x!_i − x_i)
    for each particle x⃗ ∈ P with velocity v⃗ do          ▷ Mutate
        x⃗ ← x⃗ + ϵv⃗
until Best⃗ is the ideal solution or we have run out of time
return Best⃗
```

Swarm Size

- Population sizes ranging from 10 to 50 are the most common.
- It has been learned that PSO needed smaller populations than other evolutionary algorithms to reach high quality solutions

Inertia weight

- Larger value results in smoother, more gradually changes in direction (exploration)
- Smaller value allows particle to settle into the optima (exploitation)
- The inertia weight is typically set up to vary linearly from 1 to 0 during the course

$$w(t) = \overline{w} - \frac{t}{T}(\overline{w} - \underline{w})$$

t : current iteration; T : total iterations

$\overline{w}$: upper bound; $\underline{w}$: lower bound

Settings of Acceleration Constants

$$\vec{v}_i^{(t+1)} = \alpha \vec{v}_i^{(t)} + b(\vec{x}^* - \vec{x}^{(t)}) + d(\vec{x}^! - \vec{x}^{(t)})$$

$b = r \times c_1$, where c_1 is self confidence (cognition) factor

$d = r \times c_2$, where c_2 is swarm confidence (social) factor

- Full model ($c_1, c_2 > 0$)
- Cognition only ($c_1 > 0$ and $c_2 = 0$),
- Social only ($c_1 = 0$ and $c_2 > 0$)
- Selfless ($c_1 = 0$, $c_2 > 0$, and gBest $\neq$ i)

Velocity Limit

$$v^{\max} = k \times x_{\max}$$

$$0.1 \le k \le 1.0$$

- It restricts the velocity to prevent oscillation
- This **does not** restrict the location to the range of [$-v^{max}$, v^{max}]

PSO vs. GA

	GA	PSO
General feature	Random search Population-based	Random search Population-based
Individual memory	None	Yes; through pBest
Individual operator	Mutation	pBest updating
Social operator	Selection Crossover	gBest
Balance Exploitation/ Exploration	Tunable	Higher *w*: Exploration Lower *w*: Exploitation

PSO vs. GA

- By their natures, **PSO seems good at continuous optimization** whereas **GA seems good at discrete problems**
- In PSO, particles neither die nor age
- PSO is considered **easier to implement** than GA