CSC03D3

Particle Swarm Optimisation*

*based on “Computational Intelligence - An Introduction, Second Edition”

by Andries P. Engelbrecht and slides by Dr J van Niekerk

UNIVERSITY OF JOHANNESBURG

Overview

UNIVERSITY OF JOHANNESBURG

Overview

- Basic PSO approaches
- Variations on the standard PSO
- An exploration of the parameter space for PSO
- Hybrid PSO algorithms

Basic Particle Swarm Optimisation

Introduction

- A swarm can loosely be described as a number of individuals or agents (generally mobile) working together and exchanging information (either directly or indirectly) to optimise a given problem *(see starling video)*.
- Swarm intelligence (or collective intelligence) refers to the emergent problem solving-behaviour from the algorithmic models of swarming.
- The behaviour of single individuals are usually very simple - but the collective effect of the behaviour can be extremely complex.

Emergence

Emergence can be defined as the process of deriving some new and coherent structures, patterns and properties (or behaviours) in a complex system. These structures patterns and behaviours come to existence without any coordinated control system, but emerge from interactions of individuals with their local (potentially dynamic) environment.

Examples from Nature

- Termites builds large nests with vast complexity.
- Dynamically allocated tasks in an ant colony, with no central control.
- Recruitment via waggle dances in bee colonies; trail following behaviour of ants.
- Flocks of birds; schools of fish.
- Predators (lionesses, wolf-packs).
- Bacteria communicate via molecules to keep track of the changing environment (quorum sensing).
- Slime mould aggregation (forming a mobile slug).

Conceptual Operation

- The swarm (similar to a population in EA) are flown through the hyper-dimensional search space in search for the optimal solution.
- Each particle (likened to the individual in an EA population) represents a potential solution.
- Particles update position based on a number of factors, including:
  - own past experience (a previous good point in the search space).
  - the success of other particles in the group.

Basic Particle Swarm Optimisation

- Let $\mathbf{x}_i(t)$ donate the position of the particle i in the search space at time interval t.
- The position of a particle is changed by adding a velocity $\mathbf{v}_i(t)$ to the current position:

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

with $\mathbf{x}_i(0) \sim U(\mathbf{x}_{min}, \mathbf{x}_{max})$.

Global Best PSO

- Each article connected with every other particle to exchange information (i.e. the neighbourhood of each particle is the entire swarm).
- The connectivity within the network resembles a star network.
- Each particle is drawn towards the current global best (this is not necessarily the *true best*).
- Tends to converge faster than other swarm algorithms – but often gets stuck in a local optimal.
- The velocity of particle i is calculated as:

$$v_{ij}(t+1) = v_{ij}(t) + c_1 r_{1j}(t)\left[y_{ij}(t) - x_{ij}(t)\right] + c_2 r_{2j}(t)\left[\hat{y}_j(t) - x_{ij}(t)\right]$$

Global Best PSO

Create and initialise an N_x dimensional swarm;
repeat
  for *each particle* $i = 1, 2, 3 \ldots N_s$ **do**
    if $f(\mathbf{x}_i) < f(\mathbf{y}_i)$ **then**
      $\mathbf{y}_i = \mathbf{x}_i$
    end
    if $f(\mathbf{y}_i) < f(\hat{\mathbf{y}})$ **then**
      $\hat{\mathbf{y}} = \mathbf{y}_i$
    end
  end
  for *each particle* $i = 1, 2, 3 \ldots N_s$ **do**
    update the velocity (based on velocity equation);
    update the position (based on update equation);
  end
until *stopping conditions are met*;

Local Best PSO

- Each particle is connected with a smaller neighbourhood - referred to as a social ring topology.
- Each particle is drawn to the localised best of the defined neighbourhood (which may possibly overlap).
- Converges slower, but quality of the solutions are often better.

Local Best PSO

The velocity of each particle is given by:

$$v_{ij}(t+1) = v_{ij}(t) + c_1 r_{1j}\left[y_{ij}(t) - x_{ij}(t)\right] + c_2 r_{2j}(t)\left[\hat{y}_{ij}(t) - x_{ij}(t)\right]$$

where $\hat{y}_{ij}(t)$ is the best position found in the neighbourhood of i for dimension j:

$$\hat{\mathbf{y}}_i(t+1) \in \left\{\mathcal{N}_i | f(\hat{\mathbf{y}}_i(t+1)) = \min\left\{f(\mathbf{x})\right\}, \forall \mathbf{x} \in \mathcal{N}_i\right\}$$

with the neighbourhood $\mathcal{N}_i$ defined by:

$$\mathcal{N}_i = \left\{\mathbf{y}_{i-n\mathcal{N}_i}(t), \mathbf{y}_{i-n\mathcal{N}_i+1}(t), \ldots, \mathbf{y}_{i-1}(t), \mathbf{y}_i(t), \mathbf{y}_{i+1}(t) \ldots, \mathbf{y}_{i+n\mathcal{N}_i}(t)\right\}$$

for neighbourhood size $n\mathcal{N}_i$.

Local Best PSO

- The neighbourhood is not based on the actual Euclidean "distances" between particles, but on particle indices:
  - It is computationally inexpensive! For "distances" between particles the Euclidean distances between all particles with the particle i must be calculated – which is $\mathcal{O}(n^2)$.
  - Better spread of information.
  - Neighbourhoods overlap, which allows information to migrate between neighbourhoods.

Create and initialize an n_x-dimensional swarm;
repeat
    for *each particle* $i = 1, \ldots, n_s$ **do**
        //set the personal best position
        if $f(\mathbf{x}_i) < f(\mathbf{y}_i)$ **then**
            $\mathbf{y}_i = \mathbf{x}_i$;
        end
        //set the neighborhood best position
        if $f(\mathbf{y}_i) < f(\hat{\mathbf{y}}_i)$ **then**
            $\hat{\mathbf{y}} = \mathbf{y}_i$;
        end
    end
    for *each particle* $i = 1, \ldots, n_s$ **do**

$$v_{ij}(t+1) = v_{ij}(t) + c_1 r_{1j}(t)[y_{ij}(t) - x_{ij}(t)] + c_2 r_{2j}(t)[\hat{y}_{ij}(t) - x_{ij}(t)]$$

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

    end
until *stopping condition is true*;

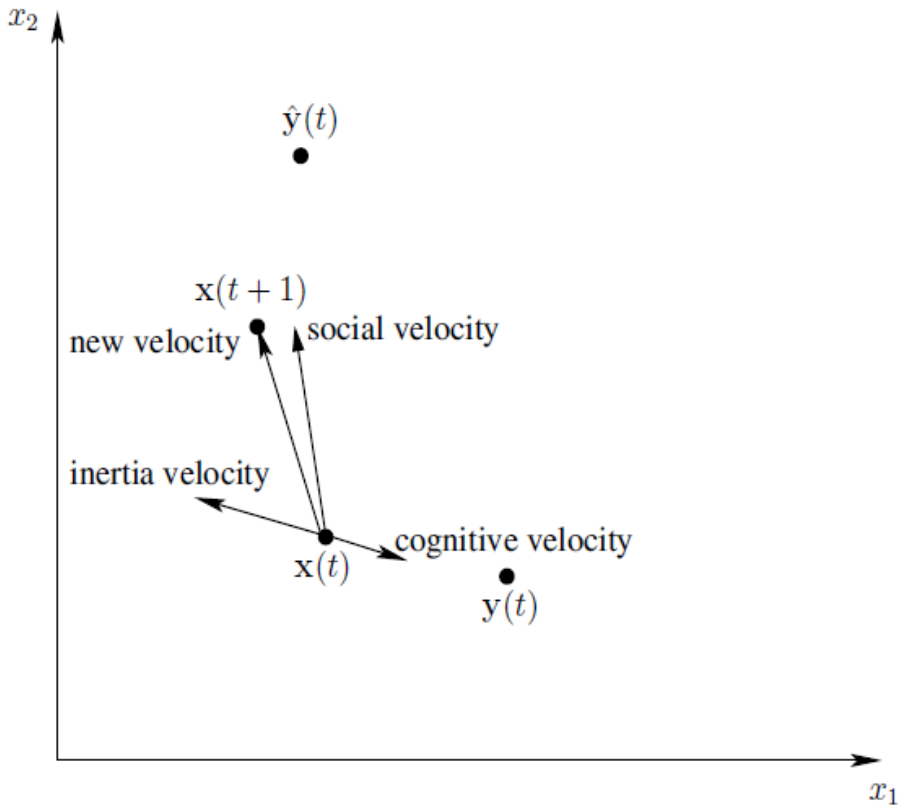
lbest vs gbest

- The *gbest* algorithm is a special case of the *lbest* algorithm, where the neighbourhood of each particle i, is the entire swarm.
- Due to larger particle interconnectivity, *gbest* tends to converge faster - however the diversity is much less than with *lbest*.
- Due to more diversity, *lbest* is less susceptible to being trapped in a local minima.

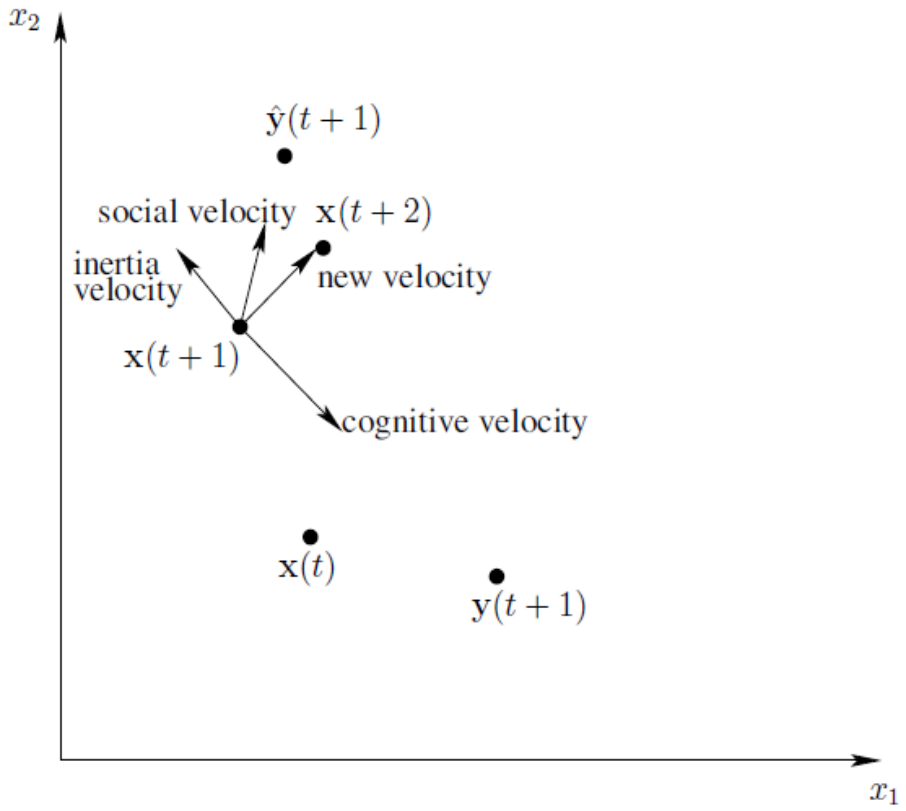
Velocity components

- The **previous velocity or inertia** which serves as the memory (or momentum) of the previous flight direction.
- The **cognitive component**, $c_1\mathbf{r}_1(\mathbf{y}_i - \mathbf{x}_i)$ quantifies the current performance of particle i based on its passed performance. It is the memory of the best position that this particle has found so far.
- The **social component**, $c_2\mathbf{r}_2(\hat{\mathbf{y}}_i - \mathbf{x}_i)$ quantifies the current performance of particle i based on performance of the swarm (or neighbourhood).

Geometric Illustration

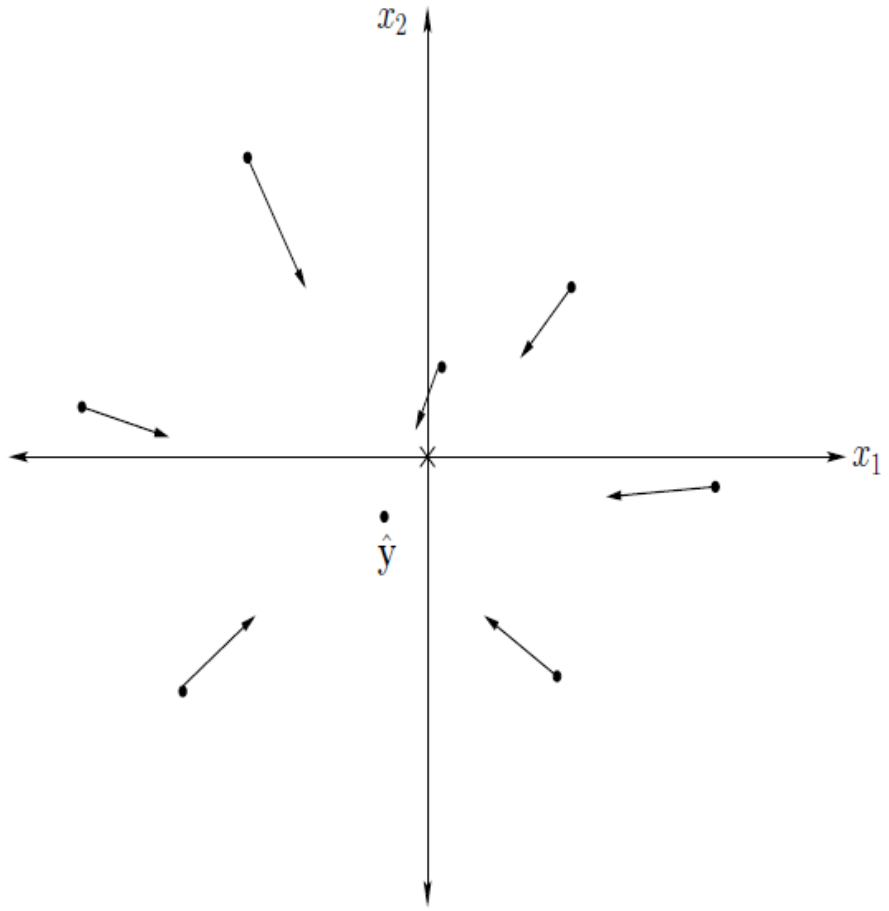


(a) Time Step t

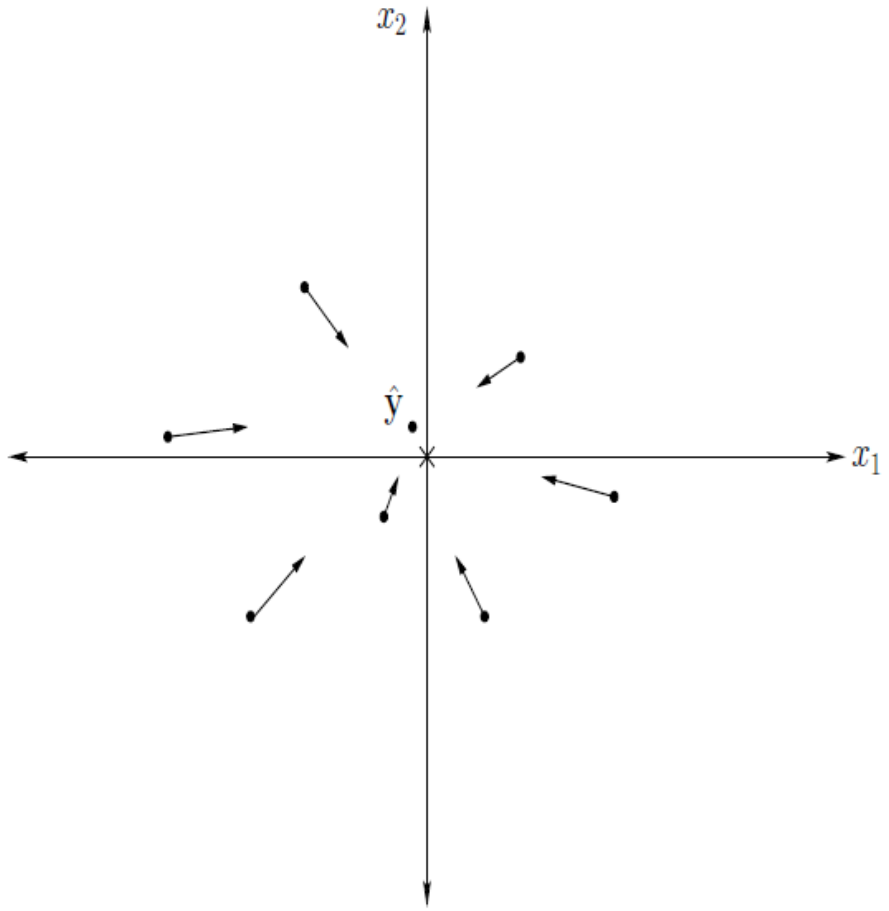


(b) Time Step $t+1$

Geometric Illustration

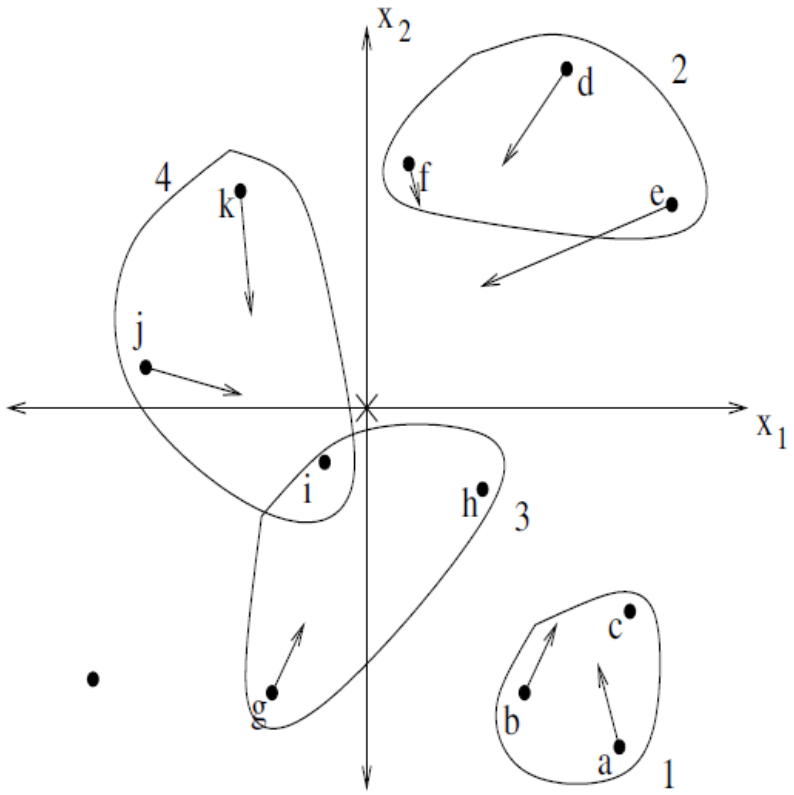


(a) At time $t = 0$

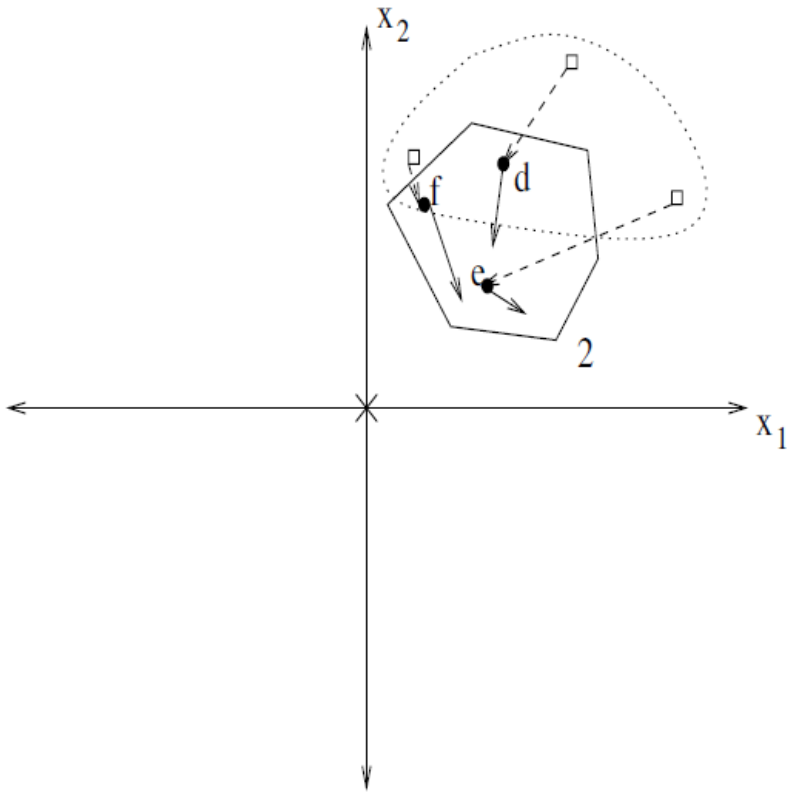


(b) At time $t = 1$

Geometric Illustration



(a) Local Best Illustrated – Initial Swarm



(b) Local Best – Second Swarm

Algorithm Aspects

- Initialisation of the algorithm:
  - $\mathbf{x}_i(0)$ is chosen randomly to cover the search space as best possible.
  - Initial velocities, $\mathbf{v}_i(0)$, are chosen to be 0.
  - Personal best position are initialised to current position, that is $\mathbf{y}_i(0) = \mathbf{x}_i(0)$.
- Stopping condition of the algorithm:
  - Should not cause the PSO to stop prematurely resulting in sub-optimal solutions.
  - The stopping condition should guard against oversampling the fitness function, resulting in additional overhead/complexity.

Termination Criteria

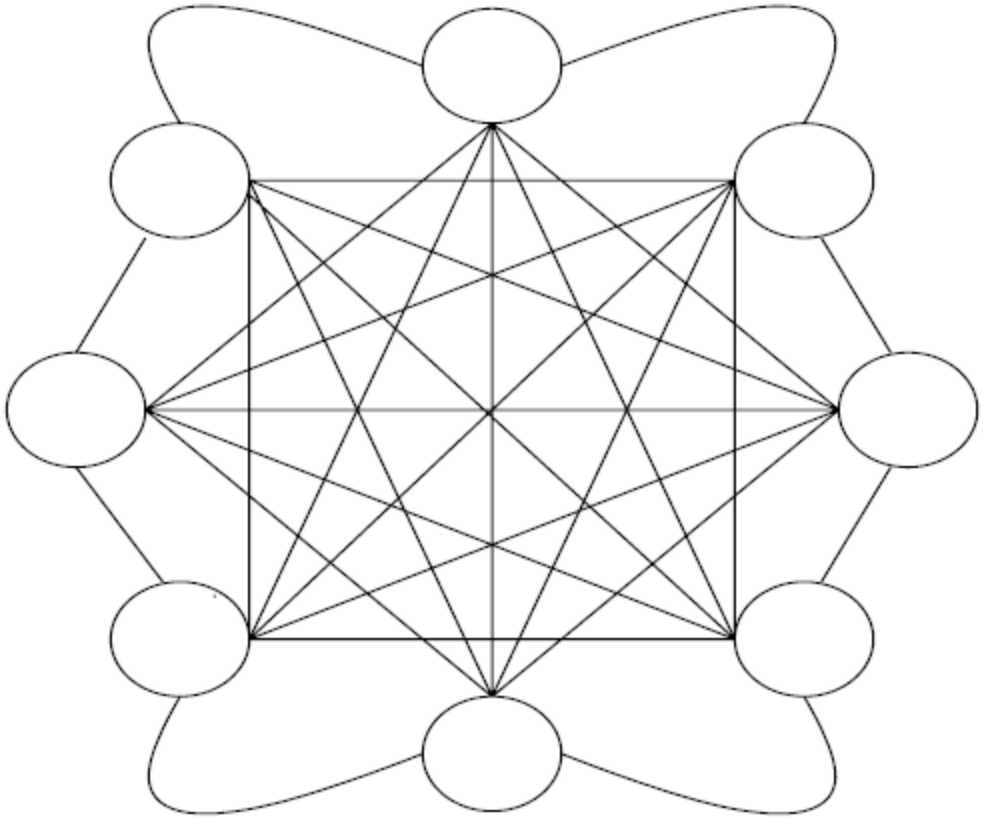
- Terminate when the a maximum number of iterations has been reached.
- Terminate after a predetermined time period.
- Terminate when an acceptable solution has been reached.
- Terminate when no improvement is observed over a number of iterations.
- Terminate when the normalised swarm radius is close to zero.
- Terminate when the objective function slope is approximately zero, i.e.

$$f'(t) = \frac{f(\hat{\mathbf{y}}(t)) - f(\hat{\mathbf{y}}(t-1))}{f(\hat{\mathbf{y}}(t))}$$

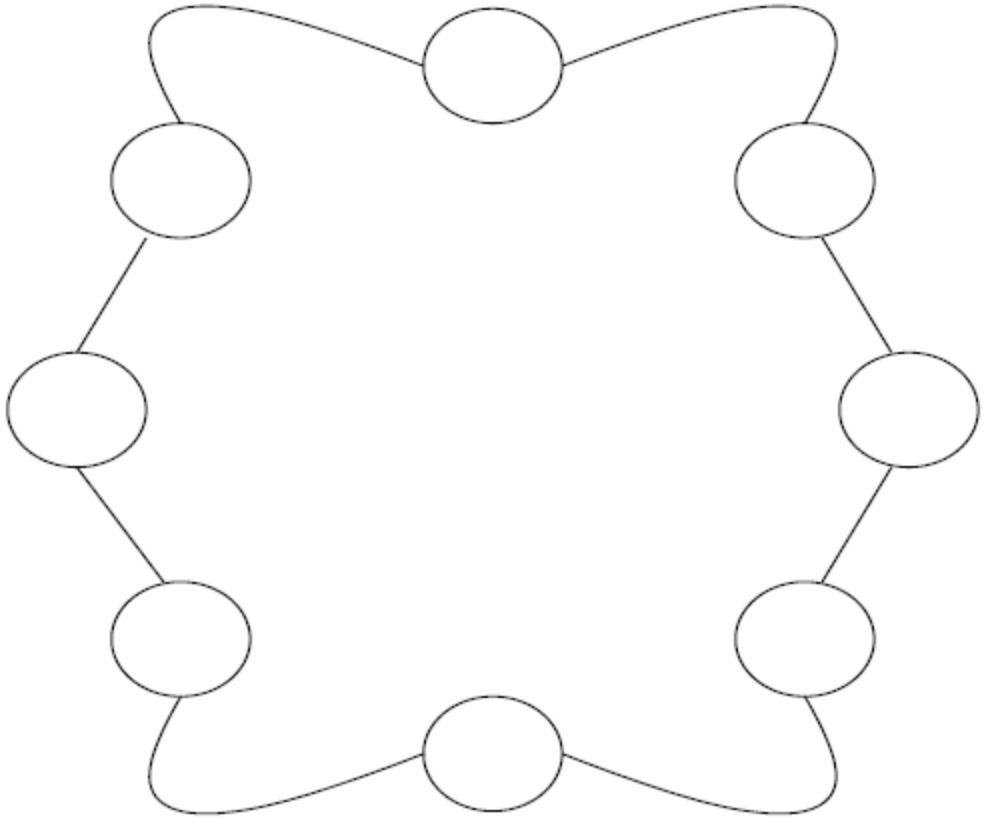
Social network structures

- **Star** – fully interconnected; quick convergence; may get stuck in local maximum.
- **Ring** – each particle communicate with defined set of neighbours; converge slower than star, but better quality solutions.
- **Wheel** – particles are isolated and communication occurs through central particle.
- **Pyramid** – forms a three-dimensional wireframe.
- **Clusters** – particles operate per cluster and exchange information similar to a ring.
- **Von Neumann** – grid-structure; has been shown to outperform other topologies in specific problem domains.

Social network structures

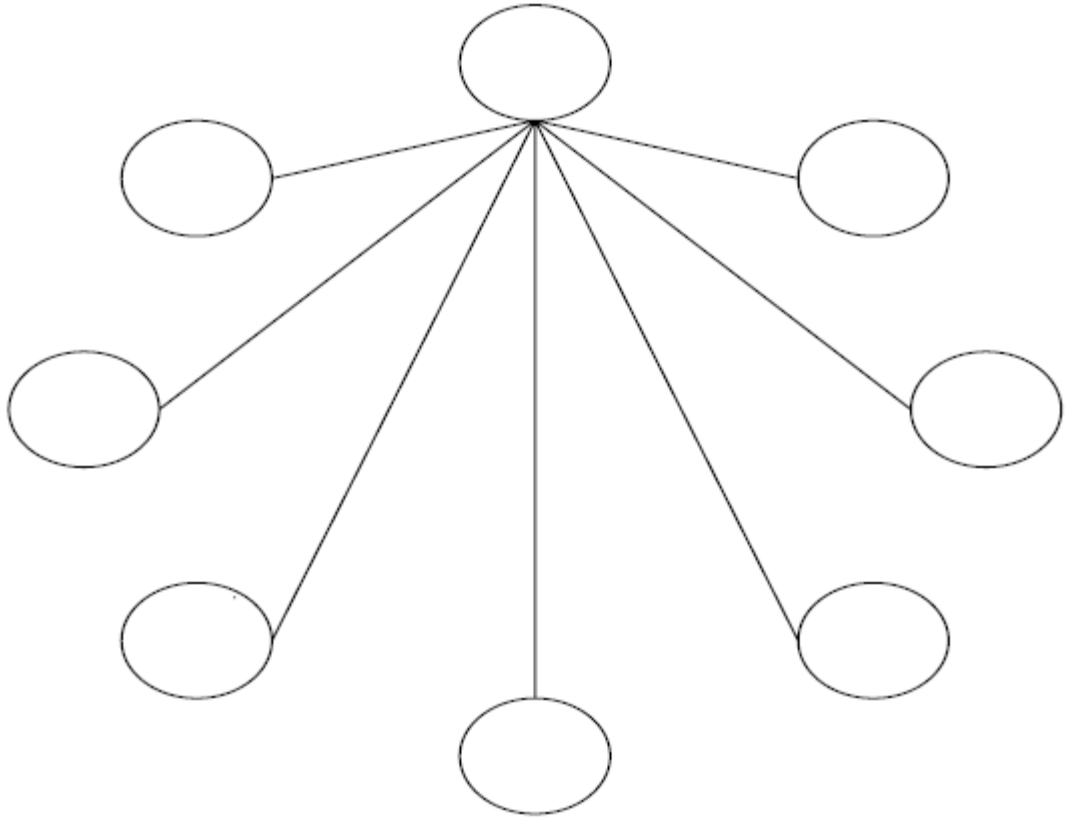


(a) Star

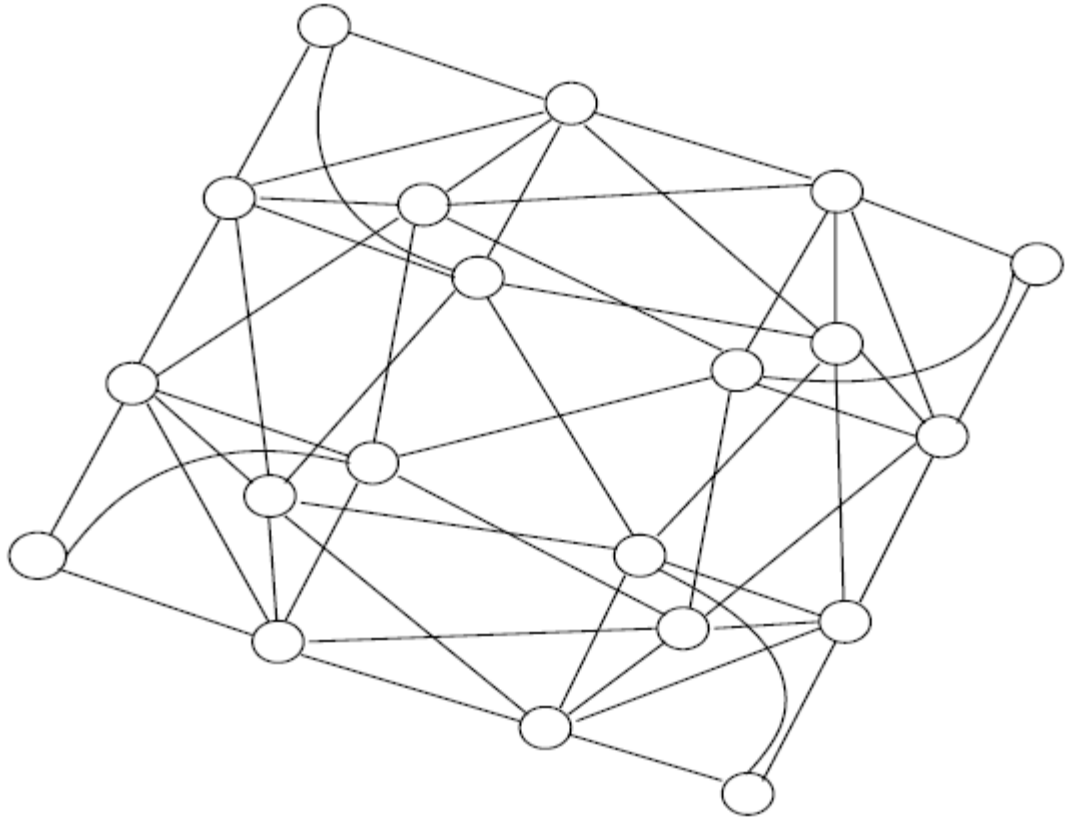


(b) Ring

Social network structures

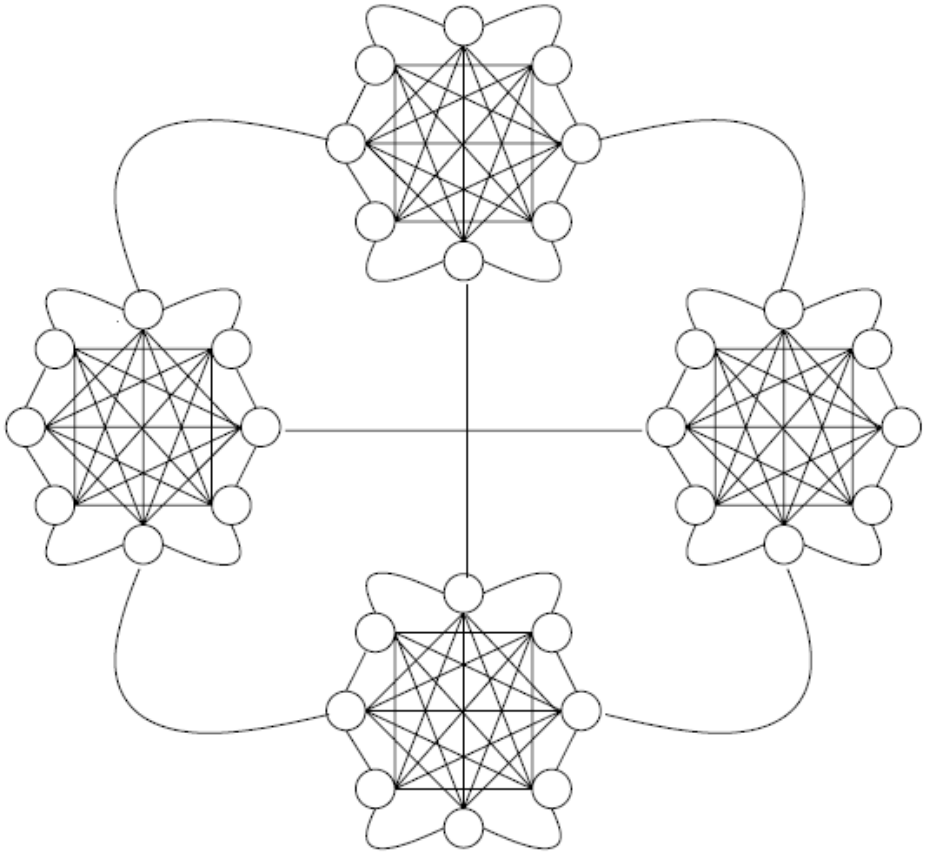


(c) Wheel

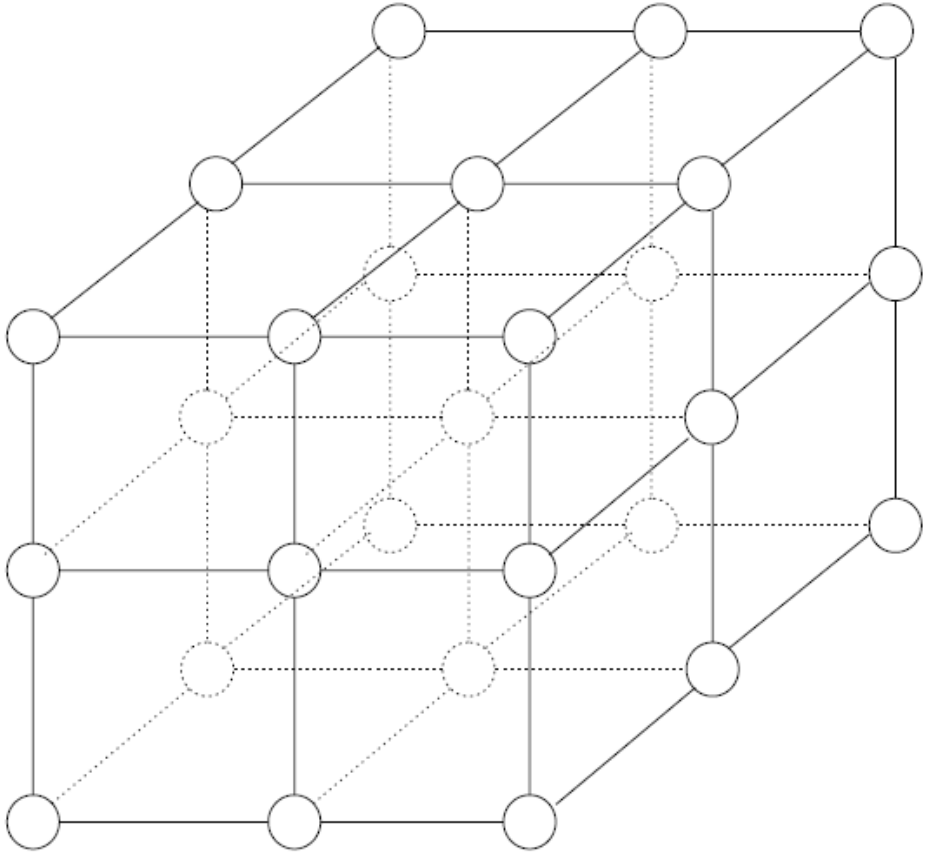


(d) Pyramid

Social network structures



(e) Four Clusters



(f) Von Neumann

Variations on Particle Swarm Optimisation

Basic Variations

- PSOs have been adapted with various "tweaks" to improve performance, including:
  - **Velocity clamping** which keeps the velocity within bounds, avoiding overshoot.
  - **Inertia weight** which attempts to balance exploration and exploitation and avoid velocity clamping.
  - **Constriction coefficient** which clamps the entire velocity in a specific way.
  - **Synchronous vs. asynchronous updates** referring to the the point in the algorithm where information is propagated through the swarm.
  - **Velocity models** which include:
    - **Cognition-only model** which ignores the social component.
    - **Social-only model** which ignores the cognative component.
    - **Selfless model** which is similar to the social model.

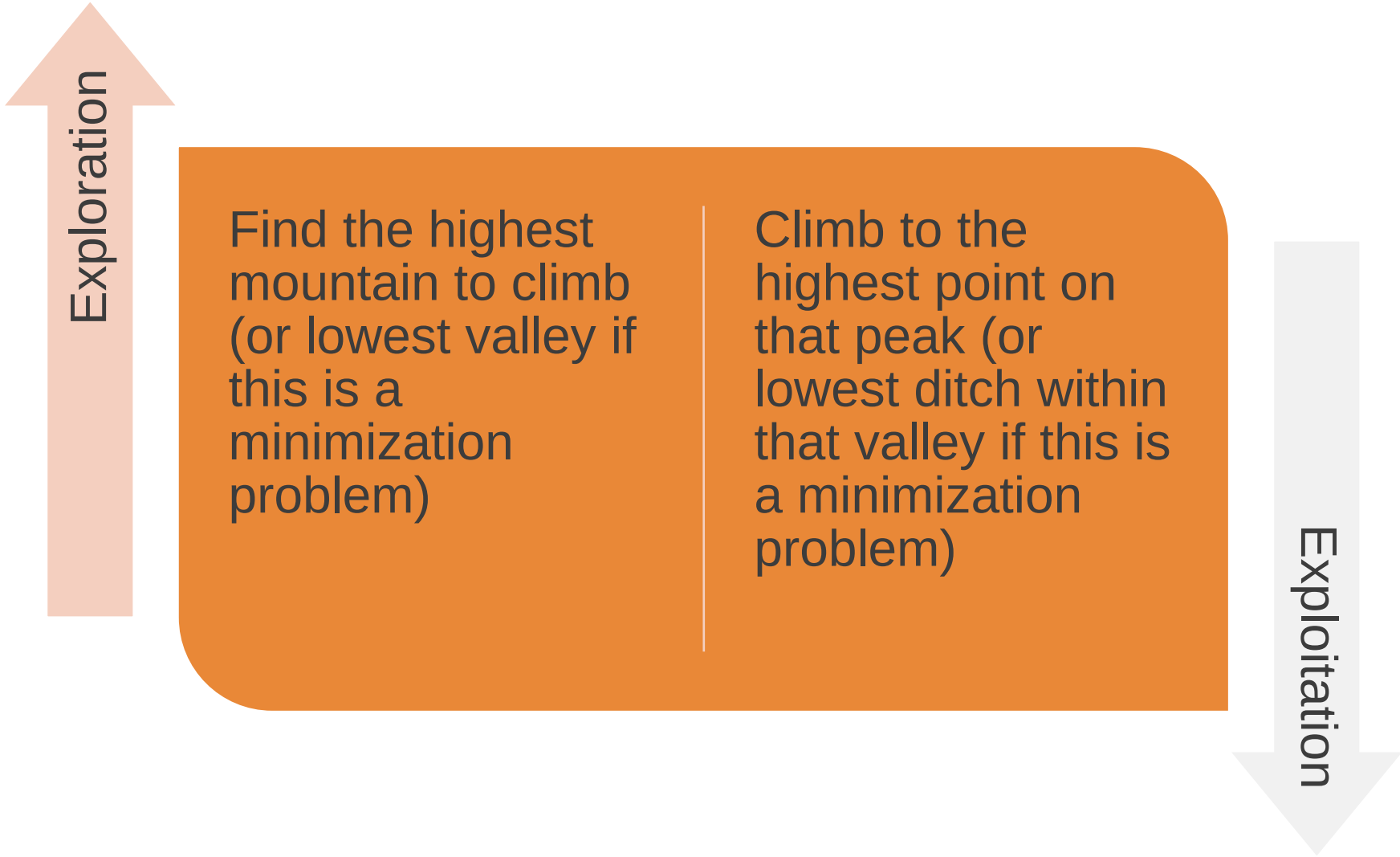
Basic Variations

- **Exploration** is the ability of a particle to explore the search space for areas that may contain promising solutions:
  - Larger velocities enhances exploration.
  - Could leave search space and overshoot good regions.
- **Exploitation** is the ability of a particle to concentrate search in an area to fine-tune a candidate solution.
  - Smaller velocities enhances exploitation.
  - Increases the chances to converge to a local optimum.

Exploration vs Exploitation

- **Exploration** is the ability of a particle to explore the search space for areas that may contain promising solutions:
  - Larger velocities enhances exploration.
  - Could leave search space and overshoot good regions.
- **Exploitation** is the ability of a particle to concentrate search in an area to fine-tune a candidate solution.
  - Smaller velocities enhances exploitation.
  - Increases the chances to converge to a local optimum.

Exploration vs Exploitation



Velocity Clamping

- Velocities can quickly grow and result in overshoots where particles will jump over large portions of the search space or out of the domain completely.
- With velocity clamping, velocities are clamped to an absolute maximum:

$$v_{ij}(t+1) = \begin{cases} v'_{ij}(t+1) & \text{if } v'_{ij}(t+1) < v_{max,j} \\ V_{max,j} & \text{if } v'_{ij}(t+1) \geq v_{max,j} \end{cases}$$

- The value of $V_{max,j}$ should be chosen with care:
  - Large values promotes exploration (but could abandon potentially good solutions).
  - Smaller values promotes exploitation (but could get stuck in local optimas).

Choosing the maximum allowed velocity

- The value of $v_{max,j}$ is very problem-specific and empirical techniques are required to find a good value. The value of $v_{max,j}$ is chosen as a fraction of each dimension, that is:

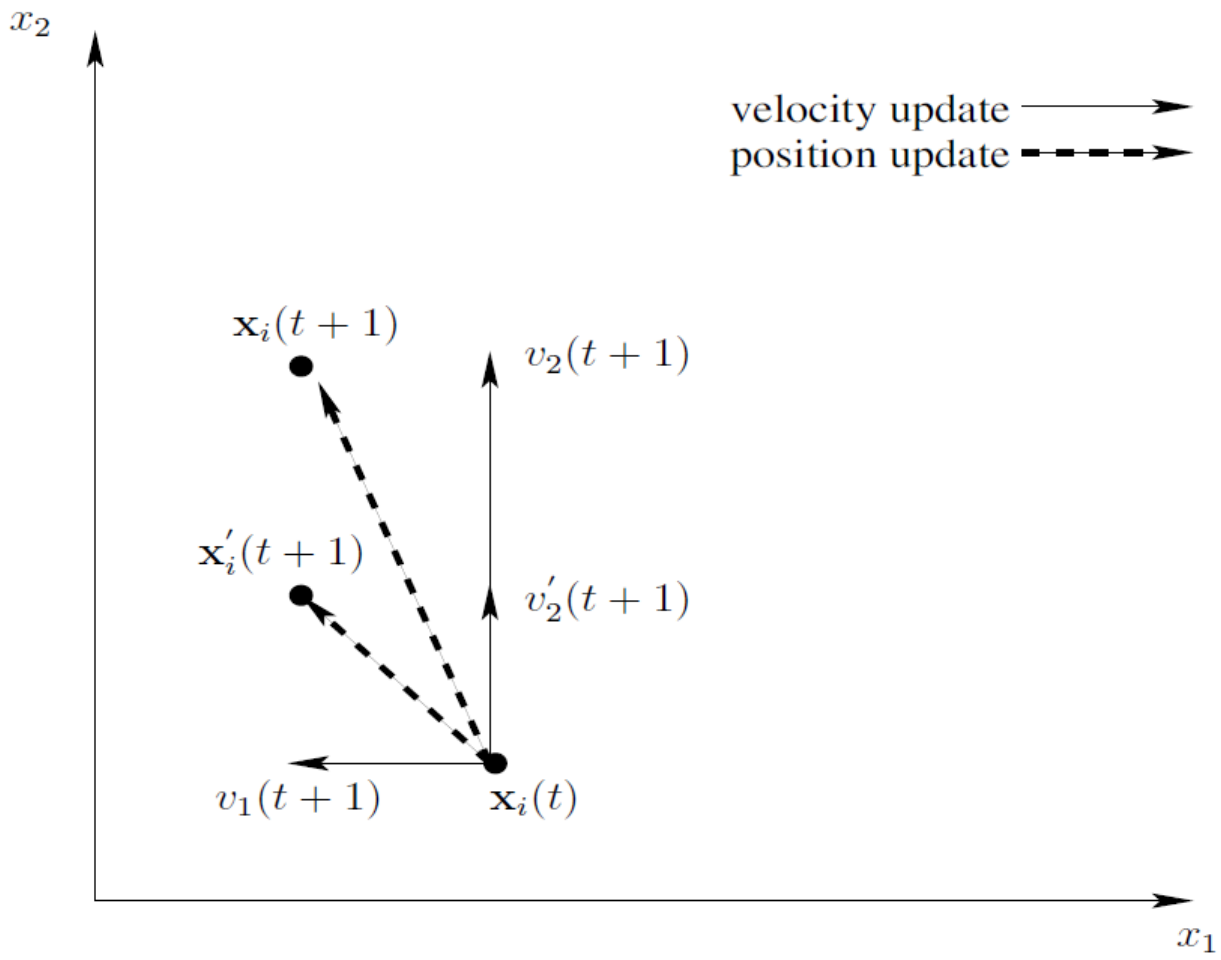
  $V_{max,j} = \delta(x_{max,j} - x_{min,j})$, where $\delta \in (0, 1]$

- Two important aspects to keep in mind:
  - Velocity clamping does not confine positions of the particles (only velocity!).
  - Velocity clamping typically restricts per dimension (i.e. specific maximum per dimension) - if the same v_{max} is used for all dimensions, then overshooting can still occur on smaller dimensions.

Consequence of non uniform velocity clamping across dimensions

- Velocity clamping per dimension changes direction
  - Can be avoided in some problems by applying velocity clamping on all dimensions simultaneously.
- Velocity clamping can result on all particles having the same (maximum) velocity, jumping from border to border. This can be avoided by
  - adding inertia weight, and
  - decreasing $V_{max,j}$ over time.

Consequence of non uniform velocity clamping across dimensions



Cooling the maximum velocity

- Change $v_{max,j}$ when no improvement is realised in the global best position over consecutive iterations (eq. 16.19).

$$v_{max,j}(t+1) = \begin{cases} \gamma V_{max,j}(t) & \text{if } f(\hat{\mathbf{y}}(t)) \geq f(\hat{\mathbf{y}}(t-t')) \quad \forall t' = 1, \ldots, n_{t'} \\ V_{max,j}(t) & \text{otherwise} \end{cases}$$

  where γ decreases from 1 to 0.01 (linearly or exponentially) using some annealing schedule.

- Exponentially decay the maximum velocity (eq. 16.20).

$$V_{max,j}(t+1) = (1 - (t/n_t)^{\alpha}) V_{max,j}(t)$$

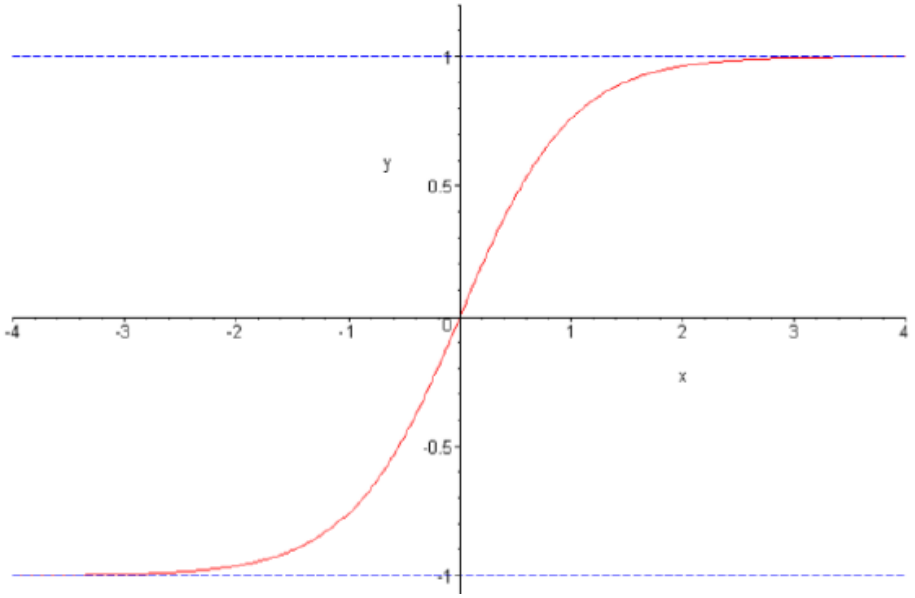
  where α is a positive constant; n_t is the maximum number of time steps (or iterations).

Stabilizing particle velocity

- v_{ij} is very sensitive to the value of δ. Some stability (and predictability) can be introduced by using the hyperbolic tangent function, i.e.

$$v_{ij}(t+1) = V_{max,j} \, tanh\left(\frac{v'_{ij}(t+1)}{V_{max,j}}\right)$$

where $v'_{ij}(t+1)$ is calculated in the usual manner (i.e. gbest or lbest).



Inertia weight

- Very simple and straightforward technique to balance exploration and exploitation, which works very well.
- Was introduced to control the exploration and exploitation abilities of the swarm and as a mechanism to eliminate velocity clamping (to some degree).

$$v_{ij}(t+1) = wv_{ij}(t) + c_1 r_{1j}\left[y_{ij}(t) - x_{ij}(t)\right] + c_2 r_{2j}(t)\left[\hat{y}_j(t) - x_{ij}(t)\right]$$

- Value of w is important:
  - For $w \geq 1$, velocities increases over time (direction changes slowly).
  - For $w < 1$, velocities decrease over time (allows quick directional changes).

Selecting w

- Optimum value for w is very problem-specific.
- Initial experiments used static values, but dynamically changing values (usually decreasing values) were used later.
- Inertia weight must be chosen in close conjunction with c_1 and c_2 (see eq. 16.23) to avoid divergent or cyclic behaviour.
- Van den Bergh and Engelbrecht showed that $w > \frac{1}{2}(c_1 + c_2) - 1$.

Dynamic adjustments to w

- Dynamically changing values (decreasing values) of w can be grouped as follows:
  - Random adjustments.
  - Linear decreasing.
  - Nonlinear decreasing.
  - Fuzzy adaptive inertia.
  - Increasing inertia.
- Linear and nonlinear update methods are similar to the temperature of simulated annealing.

Constriction Coefficient

- Very similar to inertia weight approach, but with the following differences:
  - Velocity clamping is not necessary for the constriction model.
  - Convergence is guaranteed.
  - ϕ_1 and ϕ_2 needs to be manipulated to affect any change in direction of a particle.

The velocity equation becomes:

$$v_{ij}(t+1) = \chi\left[v_{ij}(t) + \phi_1\left(y_{ij}(t) - x_{ij}(t)\right) + \phi_2\left(\hat{y}_j(t) - x_{ij}(t)\right)\right]$$

with

$$\chi = \frac{2\kappa}{|2 - \phi - \sqrt{\phi(\phi-4)}|}$$

with $\phi = \phi_1 + \phi_2$, $\phi_1 = c_1 r_1$, $\phi_2 = c_2 r_2$, $\phi \geq 4$ and $\kappa \in [0,1]$ ($\kappa \approx 0$ for exploitation and $\kappa \approx 1$ for exploration).

Synchronous vs Asynchronous updates

- Synchronous updates are done separately from the positional updates of the particles (like 'gbest' and 'lbest' presented earlier).
- Asynchronous updates, updates the global (neighbourhood) best after each particle update:
  - Provides immediate feedback to the population.
  - Should be considered in conjunction with other constants, techniques as increased information could hasten convergence.

Synchronous vs Asynchronous updates

- The 'global best' (social component):
  - The global best is usually selecting the absolute best from the neighbourhood, which can potentially be a bad solution which will then drive the swarm to a potentially bad solution.
  - The "best" position can be selected from a random best between all particles (changing periodically). This will favour exploration.
- The 'personal best' to set the 'global best':
  - Can be selected from the current position ($x_{ij}(t)$) - will favour exploration.
  - ...or from overall best found by the particle ($y_{ij}(t)$) - will favour exploitation.

Velocity Models

- Cognition-only model ("Everyone follows their own head").
  - Excludes the social component from the original velocity equation.
  - Poor algorithm.
- Social-only model ("Copycat").
  - Excludes the cognitive component from the original velocity equation.
  - Faster and slightly better than cognition-only and full model.
- Selfless model.
  - Excludes own best from the neighbourhoods best.
  - Poor performance in dynamically changing environments - but better than social-only model on certain problems.

Basic PSO Parameters

- Swarm size.
  - Problem dependent.
  - Generally, a smoother search space will require fewer particles.
- Neighbourhood size.
  - Can be static in size, but can also change;
  - Various techniques in determining neighbourhood.
- Number of iterations.
  - Problem dependent.
  - Depends on time required for a single iteration.
- Acceleration coefficients.
  - Selection of c_1 and c_2 (acceleration coefficients) and r_1 and r_2 controls the stochastic influence of the cognative/social components.

Single Solution PSOs

- Improvements have been made on the vanilla PSO obtaining better results under certain conditions, these include:
  - Guaranteed convergence PSO (GCPSO).
  - Social-based PSO.
    - Spatial social networks.
    - Fitness-based spatial neighbourhoods.
    - Growing neighbourhoods.
    - Hypercube structure.
    - Fully-informed PSO.
    - Barebones PSO.
  - Hybrid algorithms.
    - Selection-based PSO.
    - Reproduction.
    - Gaussian mutation.
    - Cauchy mutation.
    - Differential evolution based PSO.

Scaling Factor

- The scaling factor determines the radius of the search area, and is adopted using:

$$\rho(t+1) = \begin{cases} 2\rho(t) & \text{if } \#\text{successes(t)} > \epsilon_s \\ 0.5\rho(t) & \text{if } \#\text{failures(t)} > \epsilon_f \\ \rho(t) & \text{otherwise} \end{cases} \quad (3)$$

- Where #successes increments every iteration on an improvement and #failures increments if no improvement is made.

Social based PSOs

- Spatial social networks.
  - Euclidian neighbourhoods.
  - Neighbourhoods change dynamically with each iteration.
- Fitness-based spatial networks.
  - Particles groups around good particles ("hero"-based)
- Growing neighbourhoods.
  - Expanding neighbourhood.
- Hypercube structure.
  - Particles are grouped based on their hamming distance.

Social based PSOs

- Fully-informed PSO.
  - Not only the best particle is considered, but all the best positions by all the particles as the social component.
- Barebones PSO.
  - Particles converge between social and cognative component.
  - Velocity replaced by Gaussian distribution.

Hybrid Algorithms

- Hybrid algorithms.
  - Selection-based PSO.
    - Replaces the worst sub-set (typically half) with the better positions of the other sub-set, keeping $\hat{y}_j(t)$.
  - Reproduction.
    - A particle has the ability to "reproduce", kill itself or modify its parameters randomly.
  - Gaussian mutation.
    - Randomly adapt position/velocity components using a Gaussian distribution.
  - Cauchy mutation
    - Randomly adapt position/velocity components using a Cauchy mutation.
  - Differential evolution based PSO.
    - Based on genetic algorithms.
    - Three parents are randomly selected and an offspring is generated which will only replace the parent if it is better.

Multi-phase PSOs

- Multi-phase PSOs have different phases where the behaviour of each sub-swarm often changes over time or as a response to the changing environment.
- Configurations where particles migrate between groups are also possible.

Multi-phase PSOs

- Proposed by Al-Kazemi/Mohan, where particles are randomly assigned to one of two groups of equal size.
- Each sub-swarm can be in one of two states:
  - A swarm can be in an **attraction phase** (moving towards global best) or **repulsion phase** (moving away from global best).
  - The sub-swarms switch phases after set number of iterations or when no further improvements are made.

Multi-phase PSOs

- The multi-phase PSO updates its velocity as follows (note personal best missing):

$$v_{ij}(t+1) = wv_{ij}(t) + c_1 x_{ij}(t) + c_2 \hat{y}_j(t)$$

- If the tuple (w, c_1, c_2) defines the coefficients of the above equation, then:
  - Setting $(w, c_1, c_2) = (1, -1, 1)$ will force particles to move toward the global best.
  - Setting $(w, c_1, c_2) = (1, 1, -1)$ will force particles to move away from the global best.

Multi-phase PSOs

- Velocity updates periodically (decreases in frequency) reinitialised.
- Cooperation between subgroups are achieved through the selection of the global best particle.
- Particle updates follows an adapted hill-climbing approach where v consecutive components selected and added to position. New position is only accepted if it improves the particle's position.
- (Also see ARPSO, DLPSO and LCPSO on pages 328 - 328, which is self-study.)

Cooperative Split PSO

- Each particle in the swarm is split into K (split factor) separate parts of smaller dimension and each part is optimised using a separate sub-swarm.
- Fitness function becomes very difficult, as fitness of particles in sub-swarm, S_k, cannot be calculated in isolation.
- This is solved by keeping a *context vector* for fitness evaluations.
  - One method to construct the context vector is to concatenate the global best particles of each sub-swarm S_1 to S_n. To evaluate particle i, which is part of sub-swarm S_k, it merely replaces the global best particle, S_k in the current position of particle i.
  - The evaluated fitness is then assigned to that particle.
- It could happen that some sub-particles are moving towards the optimum, while others are moving away. Can be solved by optimising the sub-parts separately (however, this can prove difficult in some implementations).

Cooperative Split PSO as a parallel universe

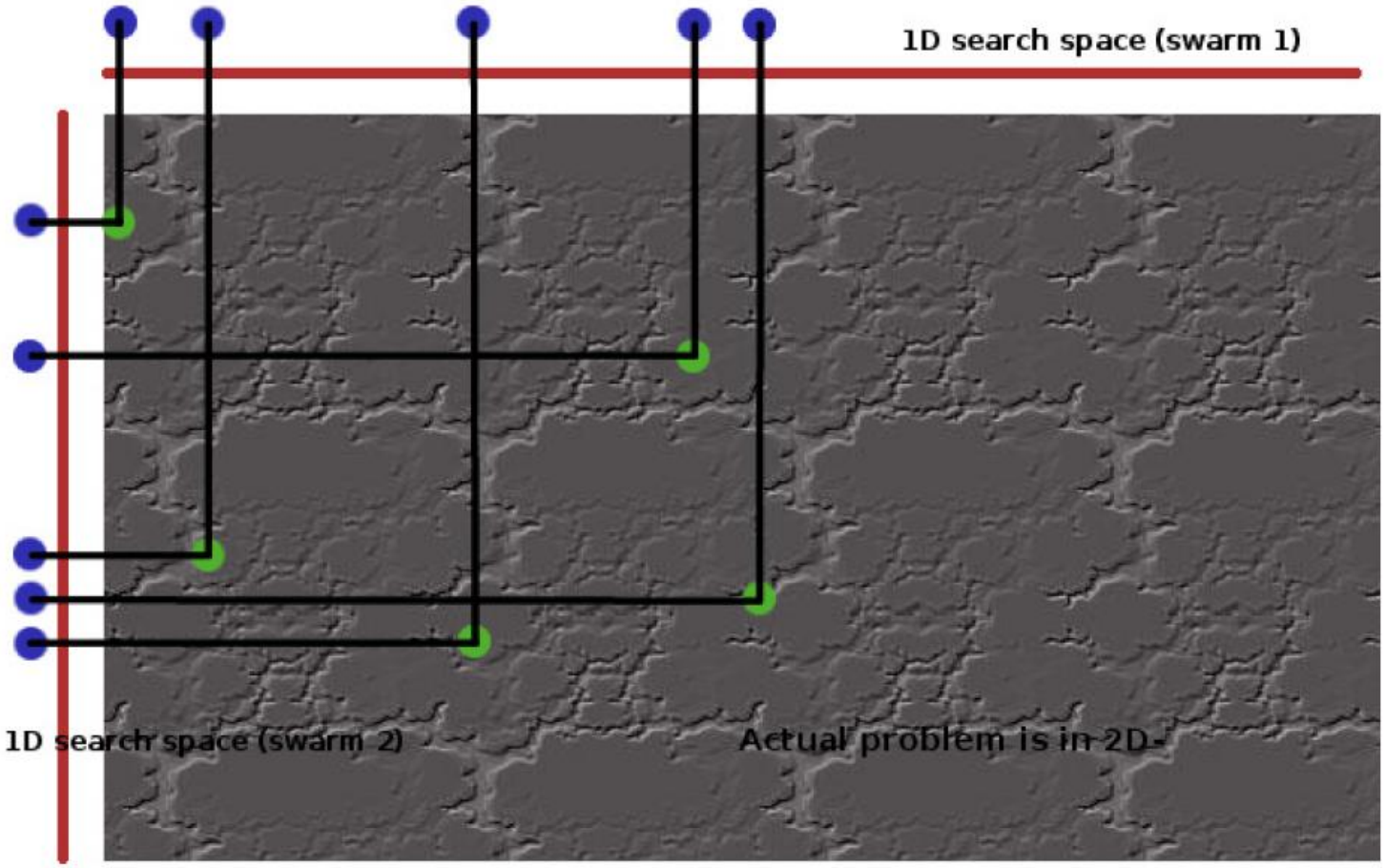
- The CSPSO can be compared to the parallel universe hypothesis.
- This is similar to the multi-dimension setup where all particles co-exist, without "knowledge" of its partner in other dimensions.

Cooperative Split Algorithm

$K_1 = n_x \bmod K$;
$K_2 = K - (n_x \bmod K)$;
Create and initialise K_1 $\lceil n_x/K \rceil$-dimensional swarms;
Create and initialise K_2 $\lfloor n_x/K \rfloor$-dimensional swarms;
repeat
  for *each sub-swarm* $S_k, k = 1, \ldots, K$ **do**
    for *each particle* $i = 1, \ldots, S_k.n_s$ **do**
      if $f(\mathbf{b}(k, S_k.\mathbf{x}_i)) < f(\mathbf{b}(k, S_k.\mathbf{y}_i))$ **then**
        $S_k.\mathbf{y}_i = S_k.\mathbf{x}_i$;
      end
      if $f(\mathbf{b}(k, S_k.\mathbf{y}_i)) < f(\mathbf{b}(k, S_k.\hat{\mathbf{y}}_i))$ **then**
        $S_k.\hat{\mathbf{y}}_i = S_k.\mathbf{y}_i$;
      end
    end
    Apply velocity and positional updates;
  end
until *until stopping conditions are met*;

Cooperative Split PSO

- Consider the problem of searching for a optimum in some 2D search space, i.e. $f(x, y)$. This can be illustrated diagrammatically as:



Predator-Prey

- It is specifically designed to balance the exploration/exploitation trade-off.
- A second swarm (usually smaller than the prey swarm) of predator particles are introduced which "scatter" prey particles which are in close proximity to a predator particle.
- Silva *et al.* used a single predator particle to pursue the global best prey particle. The velocity of the single predator particle becomes:

$$\mathbf{v}_p(t+1) = r(\hat{\mathbf{y}}(t) - \mathbf{x}_p(t))$$

where $\mathbf{v}_p$ and $\mathbf{x}_p$ are the velocity and position of the predator particle and r controls the speed with which the predator catches up with the prey.

Predator-Prey

- The prey particles updates their velocities using:

$$\mathbf{v}_{ij}(t+1) = wv_{ij}(t) + c_1 r_{1j}(t)(y_{ij}(t) - x_{ij}(t)) + c_2 r_{2j}(t)(\hat{y}_{ij}(t) - x_{ij}(t)) + s_j c_3 r_{3j}(t) D(d)$$

  where d is the Euclidean distance between prey particle i and the predator. $D(d) = \alpha e^{-\beta d}$ quantifies the influence that the predator has on the prey (which grows exponentially). (**Adendum** the value of s_j is randomly -1 or 1 so particles search around predator.)
- Components of the position vector of a particle is updated using the above equation based on a "fear" probability, P_f.
- If $U(0,1) < P_f$ then the above equation is used, otherwise the standard PSO equation is applied.

Multi-Start PSOs

- When the basic PSO converges, a lack of diversity occurs.
- The lack of diversity can be addressed by injecting randomness to the swarm to increase diversity.
  - Difference between chaos/stochastic properties and true diversity.
  - The injection of randomness increases the negative entropy in the swarm
  - Continuous random injection will result that the swarm never reach stable equilibrium.
- When/which components/particles are "tweaked" should be carefully considered.

Multi-Start PSOs

- If **positions** are randomized, particles are "teleported" to different areas, while if **velocity vectors** are randomized, particles are "thrown" of course to seek out different areas.
  - In both cases "personal" and "global" best positions are kept.
  - If no improvement is found, particles will again move towards previous best positions.

Multi-Start PSOs

- Total reinitialisation also possible at specific intervals. Number of considerations:
  - If "personal best" is also reset so that particles are initially only drawn towards "global best".
  - Velocities usually set to zero, but can also be initialised toward old cognative component (before being reset).

Multi-Start PSOs

- The number of particles and the timing of re-initialisation very important:
  - At fixed intervals (may happen that some particles are still making progress).
  - Probabilistic approaches, where all components are reinitialised based on some probability. Initial probabilities are large and are decreased over time.
  - Approaches based on a convergence condition (i.e. no improvement over time).
- Lastly which particles are initialised:
  - The global best particle is usually **not** reset to a different value.
  - Only a subsection, worse particles, particles that do not improve, etc. have been considered.

Charged PSO

- The number of particles and the timing of re-initialisation very important:
  - At fixed intervals (may happen that some particles are still making progress).
  - Probabilistic approaches, where all components are reinitialised based on some probability. Initial probabilities are large and are decreased over time.
  - Approaches based on a convergence condition (i.e. no improvement over time).
- Lastly which particles are initialised:
  - The global best particle is usually **not** reset to a different value.
  - Only a subsection, worse particles, particles that do not improve, etc. have been considered.

Charged PSO

- Attraction to the centre of the swarm and inter-particle repulsion.
- The idea is to introduce two opposing forces within the dynamics of the PSO:
  - An **attraction** force to the center of the mass of the swarm (each bird in the flock wants to be in close proximity of the other members of the flock).
  - An inter-particle **repulsion** (a bird wants to avoid collisions with other birds).

Charged PSO

- An acceleration variable to the standard velocity equation to facilitate repulsion/attraction:

$$v_{ij}(t+1) = v_{ij}(t) + c_1 r_{1j} [y_{ij}(t) - x_{ij}(t)] + c_2 r_{2j}(t) [\hat{y}_j(t) - x_{ij}(t)] + a_{ij}(t)$$

- The acceleration for each particle is calculated based on the proximity/acceleration of other particles (see eq. 16.87 and 16.88 on p. 337).
- Three models:
  - Neutral swarm (where $Q_i = 0$ for all particles, i.e. the standard PSO).
  - Charged swarm (each particle has a charge).
  - Atomic swarm (where half of the particles are charged and the other other half neutral).

Particles with Spatial Extension

- Particles with spatial extension was developed to avoid premature conversion when particles start to cluster.
- Three strategies have been investigated by Krink *et al.*:
  - **Random bouncing,** where particles move in a random new direction at the same speed as before the collision.
  - **Realistic physical bouncing,** which is based on a defined physics model.
  - **Simple velocity-line bouncing**, where a particle moves in the same direction but at a scaled speed.
- Random bouncing has been shown to be less efficient that consistent bouncing.
- It has also been empirically shown that probability bouncing (allowing some particles to converge) have a better result.
- Probability of bouncing can be decreased over time to facilitate convergence (exploitation).

Binary PSO

- Specifically designed for discrete problems (recall that PSOs were designed continuous-valued search spaces).
- Some discrete problems can be solved with the standard PSO after binary/grey coding.
- Particles in the Binary PSO represents positions in binary space. Each binary component of each particle can be either 0 or 1.
- Change in position is simply the toggling of bits (a complete toggle of all components will move the particle to opposite corner of the hypercube).

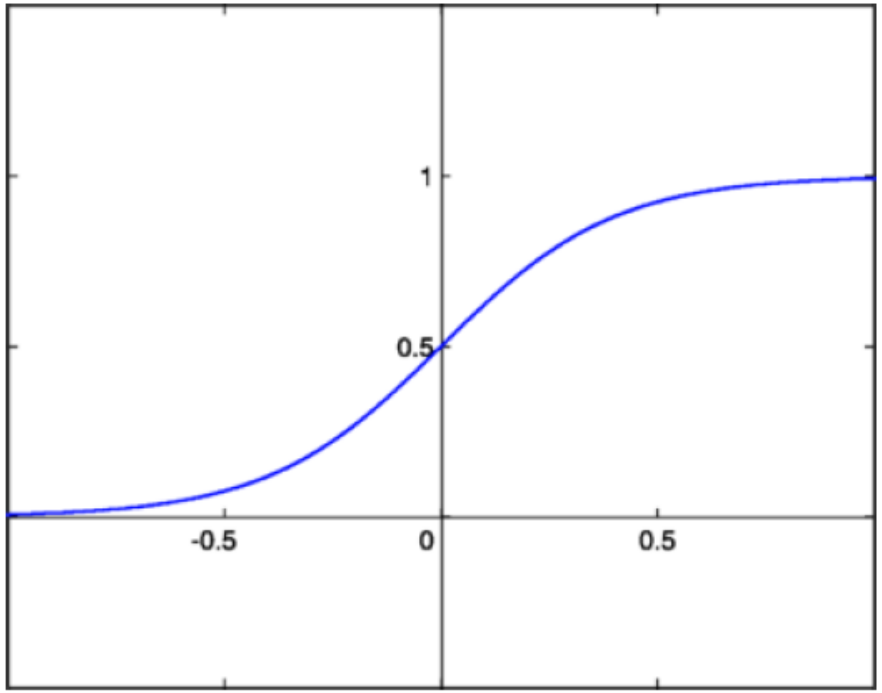
Binary PSO

- The velocity of the entire particle can be interpreted as the Hamming distance between two consecutive positions (i.e. $\mathbf{x}_i(t)$ and $\mathbf{x}_i(t+1)$.)
- On a binary level, a flip denotes the maximum velocity possible. To avoid such drastic jumps, velocity should rather be interpreted using a probability, $p \in [0, 1]$, on bit-level.
  - For example, if $v_{ij}(t) = 0.3$, then particle i has a 30% chance of being 1 and a 70% chance of being 0.
  - Velocities can be normalised by either dividing by the maximum allowed velocity or sigmoid function (better).

Binary PSO

- The velocity updates (using the sigmoid function) becomes:

$$v'_{ij}(t) = sig(v_{ij}(t)) = \frac{1}{1 + e^{-v_{ij}(t)}}$$



Binary PSO

- . . . and the position update becomes:

$$x_{ij}(t) = \begin{cases} 1 & \text{if } r_{3j} < sig(v_{ij}(t+1)) \\ 0 & \text{otherwise} \end{cases}$$

- with $r_{3j} \sim U(0,1)$. This implies that if $v_{ij}(t) = 0$ then there is exactly a 50% probability that $x_{ij}(t) = 1$. If $v_{ij}(t) < 0$ the probability is less than 50% and if $v_{ij}(t) > 0$ then the probability is greater than 50%.