AI791 (Computational Intelligence for Optimization Topic 3: Particle Swarm Optimization

AP Engelbrecht

Department of Industrial Enigneering, and
Division of Computer Science
Stellenbosch University
South Africa
engel@sun.ac.za
https://engel.pages.cs.sun.ac.za/

Presentation Outline I

1. Reading Material
2. Particle Swarm Optimization
   - Origins of Particle Swarm Optimization
   - Basic Foundations of Particle Swarm Optimization
   - Convergence Behavior of Particle Swarm Optimization
   - Particle Roaming Behaviour
   - Velocity Initialization
   - Single Solution Particle Swarm Optimization Algorithms
   - Self-Adaptive Particle Swarm Optimization
   - Heterogeneous Particle Swarm Optimization
3. Particle Swarm Optimization for Solving Different Optimization Problem Classes
   - Large Scale Optimization
   - Discrete-Valued Optimization Problems
   - Multi-Solution Optimization Problems
   - Dynamic Optimization problems

Presentation Outline II

- Constrained Optimization Problems
- Multi-Objective Optimization Problems

Reading Material

The content of this topic is from Chapter 16 of the prescribed book, *Computational Intelligence: An Introduction*, AP Engelbrecht, Second Edition, Wiley, 2007

Articles will serve as additional reading material

Topic 3: Particle Swarm Optimization

To be discussed:

- Origins of Particle Swarm Optimization
- Basic Foundations of Particle Swarm Optimization
- Convergence Behavior of Particle Swarm Optimization
- Particle Roaming Behaviour
- Velocity Initialization
- Particle Swarm Optimization Algorithms
- Self-Adaptive Particle Swarm Optimization
- Heterogeneous Particle Swarm Optimization

Particle Swarm Optimization

Origins

Particle swarm optimization (PSO):

- developed by Kennedy & Eberhart,
- first published in 1995, and
- with an exponential increase in the number of publications since then.

What is PSO?

- a simple, computationally efficient optimization method
- population-based, stochastic search
- individuals follow very simple behaviors:
  - emulate the success of neighboring individuals,
  - but also bias towards own experience of success
- emergent behavior: discovery of optimal regions within a high dimensional search space

Particle Swarm Optimization

Origins: Boids

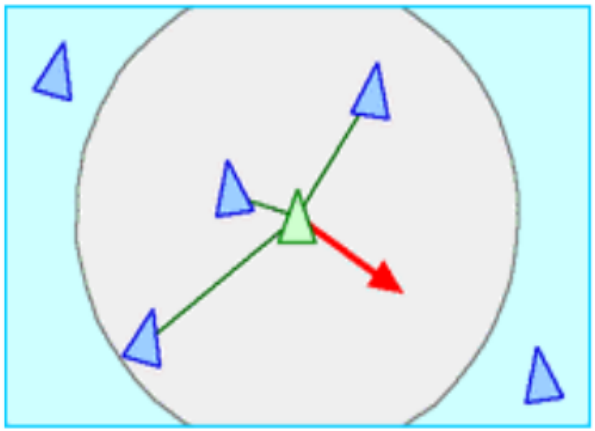
What are the origins of PSO?

In the work of Reynolds on “boids” in 1987

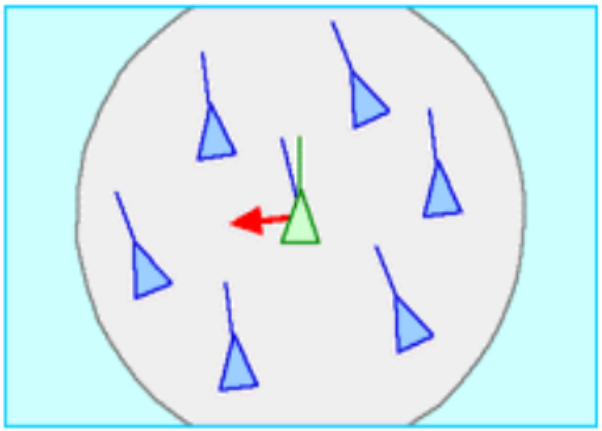
- Flocking behavior is the result of the motion and interaction of boids
- Each boid has three simple rules of steering behavior, describing how an individual boid moves based on the positions and velocities of the other boids:
  - Separation – each boid keeps a distance from other boids nearby to avoid collision and prevent crowding
  - Alignment – each boid matches the direction and the speed of its neighbours
  - Cohesion – each boid tends to move to the average position of its neighbours

Particle Swarm Optimization

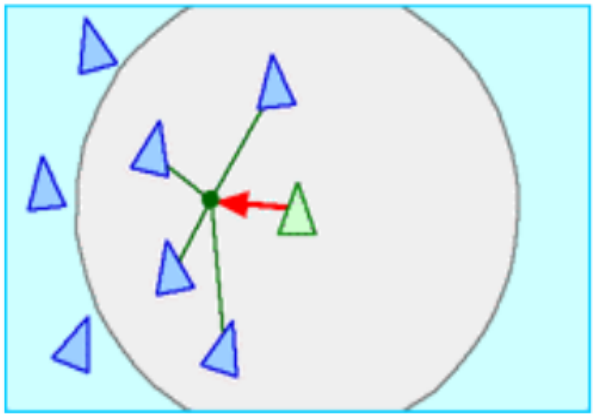
Origins: Boids (cont)



(a)



(b)



(c)

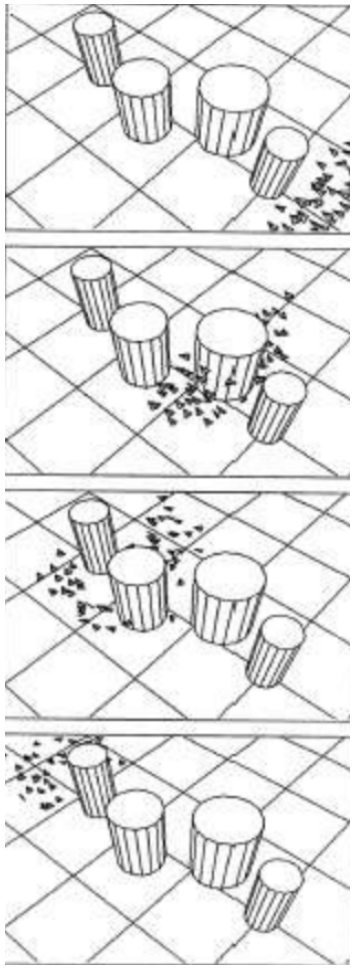
(a) Separation rule
(b) Alignment rule
(c) Cohesion rule

Particle Swarm Optimization

Origins: Boids (cont)

Reynolds adapted the boid model in 1999 to include individual-based rules of the steering behavior:

- Obstacle avoidance – obstacle avoidance behaviour allows the boids to move in cluttered environments by dodging around obstacles
- Leader following – this behaviour causes one or more boids to follow another moving boid selected as a leader



Particle Swarm Optimization

Origins: Rooster Effect

The work of Heppner and Grenander on using a “rooster” as attractor of all birds in the flock

Particle Swarm Optimization

Origins (cont)

- Original PSO is a simplified social model of determining nearest neighbors and velocity matching
- Initial objective: to simulate the graceful, unpredictable choreography of collision-proof birds in a flock
  - Randomly initializes positions of birds
  - At each iteration, each individual determines its nearest neighbor and replaces its velocity and direction with that of its neighbor
- This resulted in synchronous movement of the flock, but the flock settled too quickly on an unanimous, unchanging flying direction

Particle Swarm Optimization

Origins (cont)

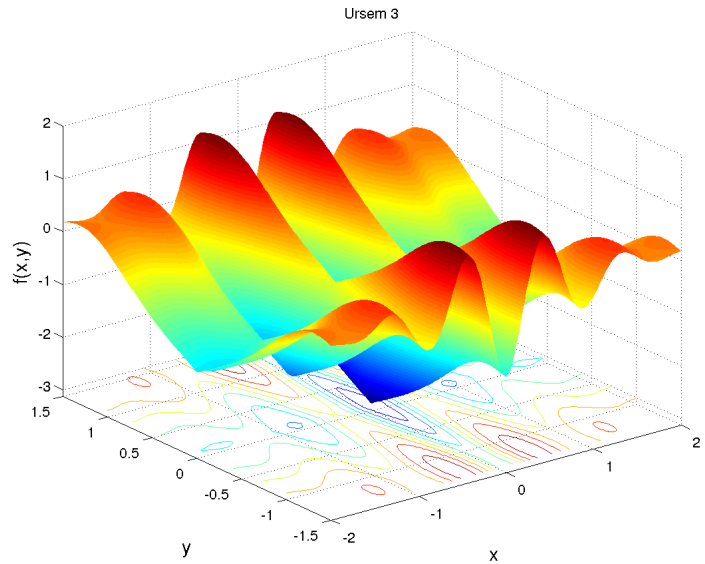
- Random adjustments to velocities (referred to as craziness) prevented individuals to settle too quickly on an unchanging direction
- To further expand the model, roosters were added as attractors:
  - personal best
  - neighborhood best

$\rightarrow$ particle swarm optimization

PSO has been developed to solve multi-dimensional optimization problems

Original PSO was developed to solve optimization problems of the following type:

- boundary constrained
- $\mathbf{x} \in \mathbb{R}^{n_x}$
- single objective function
- stationary environments



What are the main components?

- a swarm of particles
- each particle represents a candidate solution
- elements of a particle represent parameters to be optimized

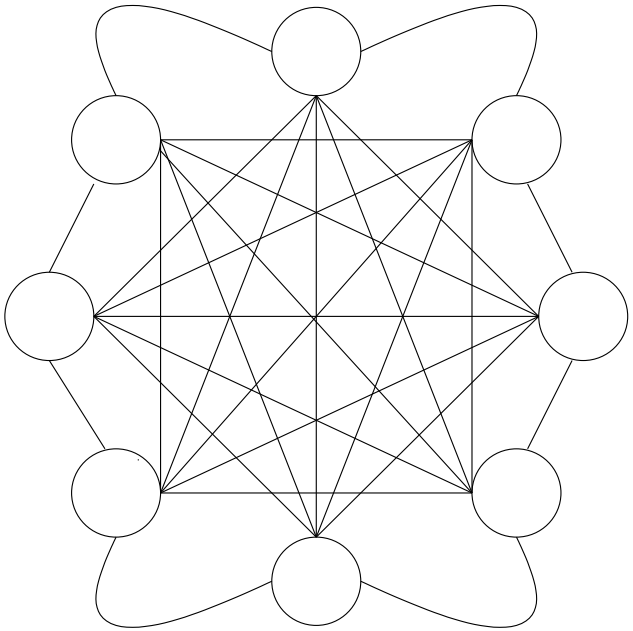
The search process:

- Position updates

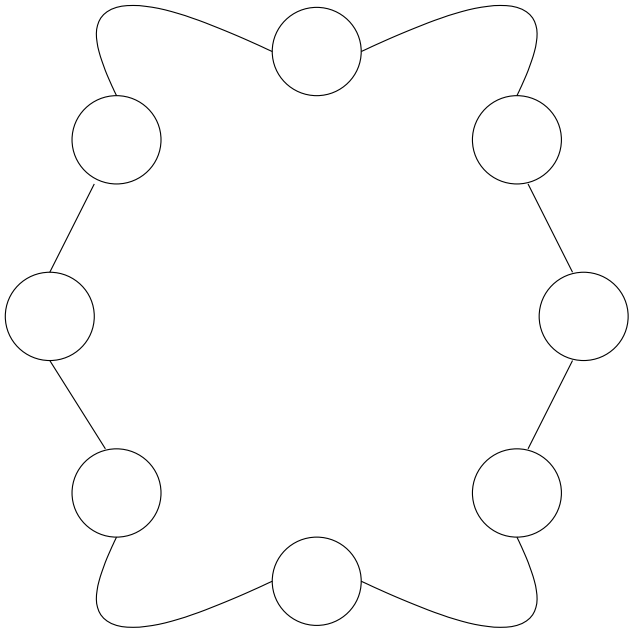
$$\mathbf{x}_i(t+1) = \mathbf{x}_i(t) + \mathbf{v}_i(t+1), \quad \mathbf{x}_{ij}(0) \sim U(x_{min,j}, x_{max,j})$$

- Velocity (step size)
  - drives the optimization process
  - step size
  - reflects experiential knowledge and socially exchanged information

Social network structures are used to determine best positions/attractors



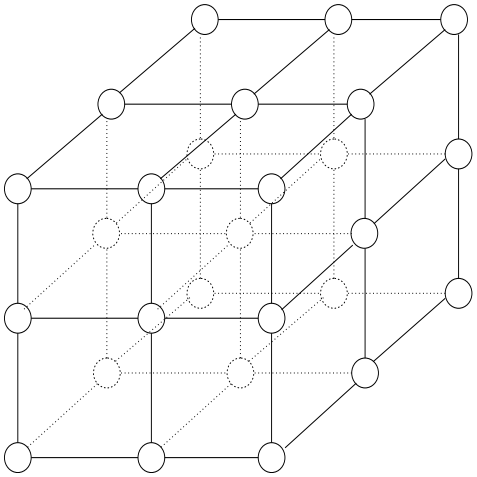
Star Topology (gbest PSO)



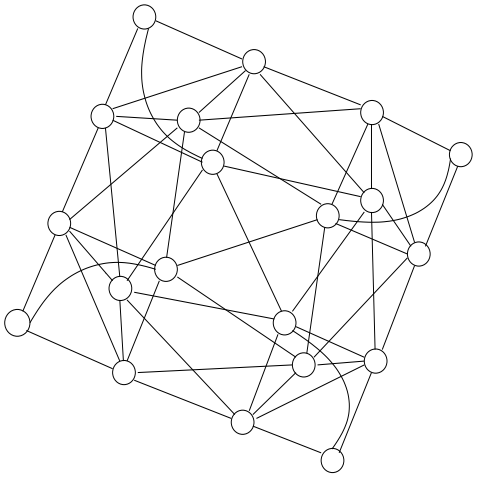
Ring Topology (lbest PSO)

Basic Foundations of Particle Swarm Optimization

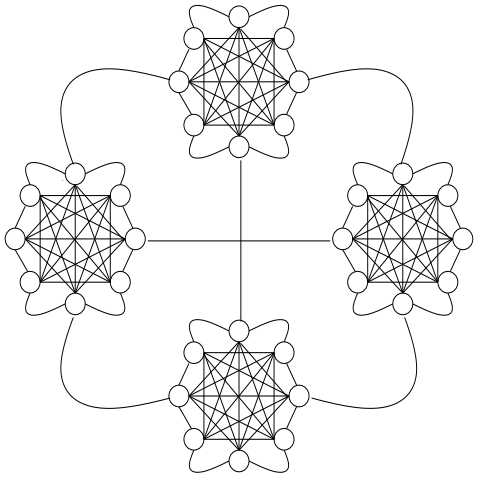
Social Network Structures (cont)



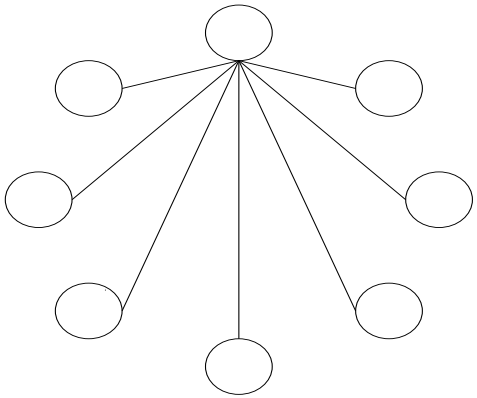
(a) Von Neumann



(b) Pyramid



(c) 4 Clusters



(d) Wheel

- uses the star social network
- velocity update per dimension:

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

- $v_{ij}(0) = 0$ (to be discussed later)
- c_1, c_2 are positive acceleration coefficients
- $r_{1j}(t), r_{2j}(t) \sim U(0,1)$
- note that a random number is sampled for each dimension (to be discussed later)

- $\mathbf{y}_i(t)$ is the personal best position calculated as (assuming minimization)

$$\mathbf{y}_i(t+1) = \begin{cases} \mathbf{y}_i(t) & \text{if } f(\mathbf{x}_i(t+1)) \geq f(\mathbf{y}_i(t)) \\ \mathbf{x}_i(t+1) & \text{if } f(\mathbf{x}_i(t+1)) < f(\mathbf{y}_i(t)) \end{cases}$$

- $\hat{\mathbf{y}}(t)$ is the global best position calculated as

$$\hat{\mathbf{y}}(t) \in \{\mathbf{y}_0(t), \ldots, \mathbf{y}_{n_s}(t)\} | f(\hat{\mathbf{y}}(t)) = \min\{f(\mathbf{y}_0(t)), \ldots, f(\mathbf{y}_{n_s}(t))\}$$

or (removing memory of best positions)

$$\hat{\mathbf{y}}(t) = \min\{f(\mathbf{x}_0(t)), \ldots, f(\mathbf{x}_{n_s}(t))\}$$

where n_s is the number of particles in the swarm

Create and initialize an n_x-dimensional swarm, S;
repeat
  for *each particle* $i = 1, \ldots, S.n_s$ **do**
    if $f(S.\mathbf{x}_i) < f(S.\mathbf{y}_i)$ **then**
      $S.\mathbf{y}_i = S.\mathbf{x}_i$;
    end
    if $f(S.\mathbf{y}_i) < f(S.\hat{\mathbf{y}})$ **then**
      $S.\hat{\mathbf{y}} = S.\mathbf{y}_i$;
    end
  end
  for *each particle* $i = 1, \ldots, S.n_s$ **do**
    update the velocity;
    update the position;
  end
until *stopping condition is true*;

- uses the ring social network

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

- $\hat{\mathbf{y}}_i$ is the neighborhood best, defined as

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

  with the neighborhood defined as

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

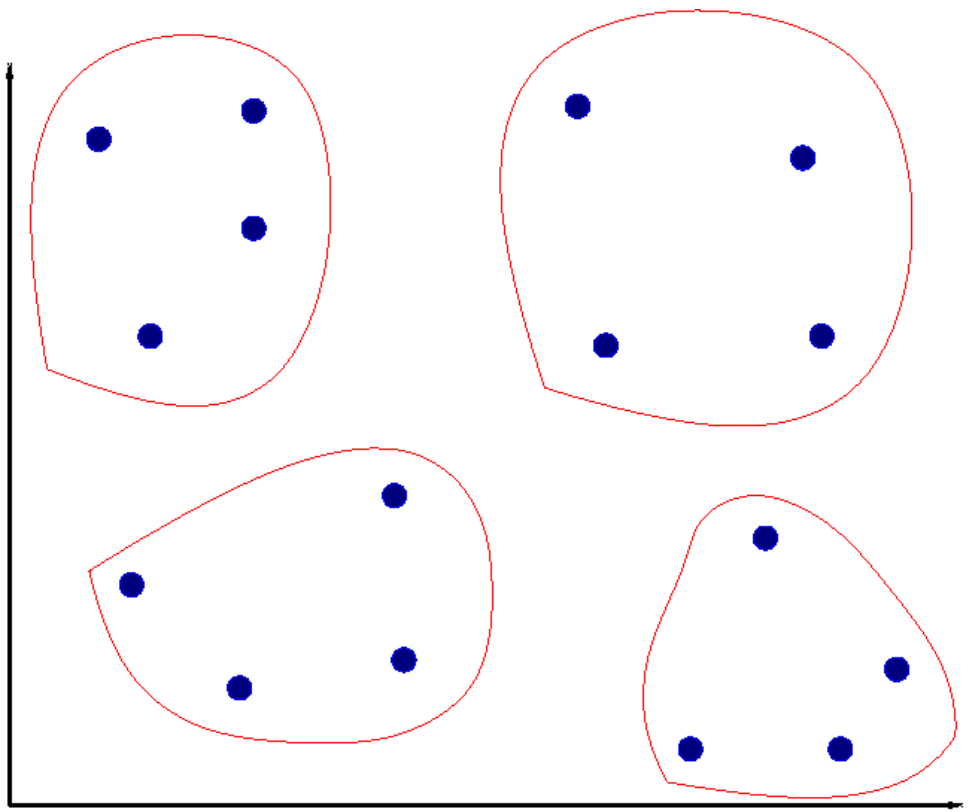
  where $n_{\mathcal{N}_i}$ is the neighborhood size
- neighborhoods overlap to facilitate information exchange

Basic Foundations of Particle Swarm Optimization

local best (lbest) PSO (cont)

Neighborhoods based on particle indices, not spatial information

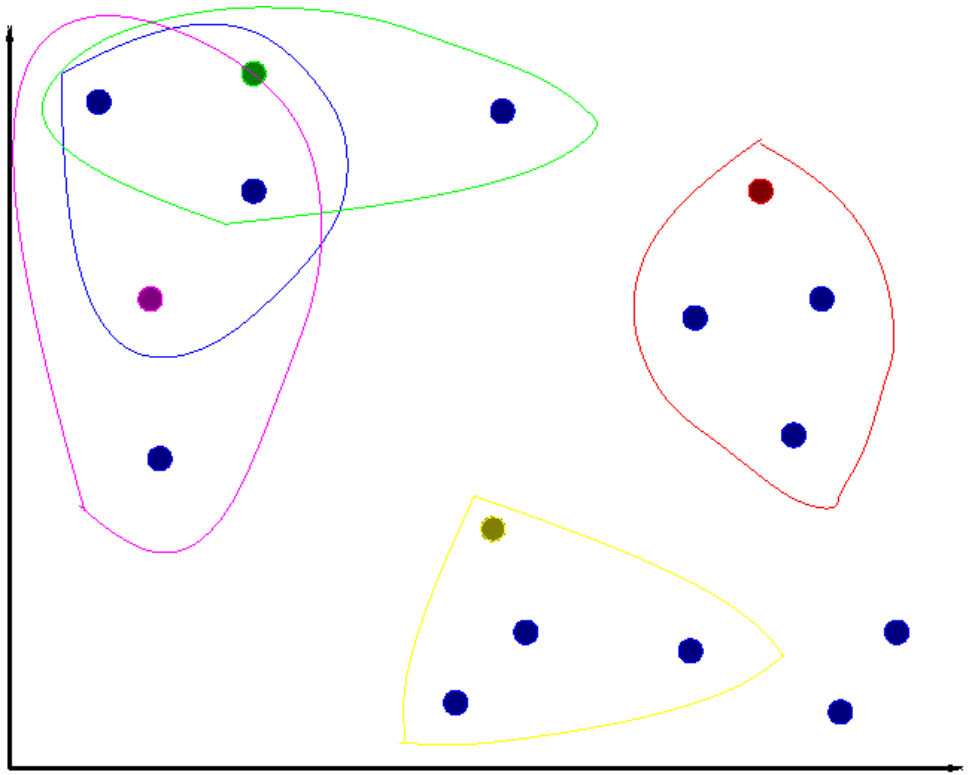
Consider non-overlapping neighbourhoods based on spatial information



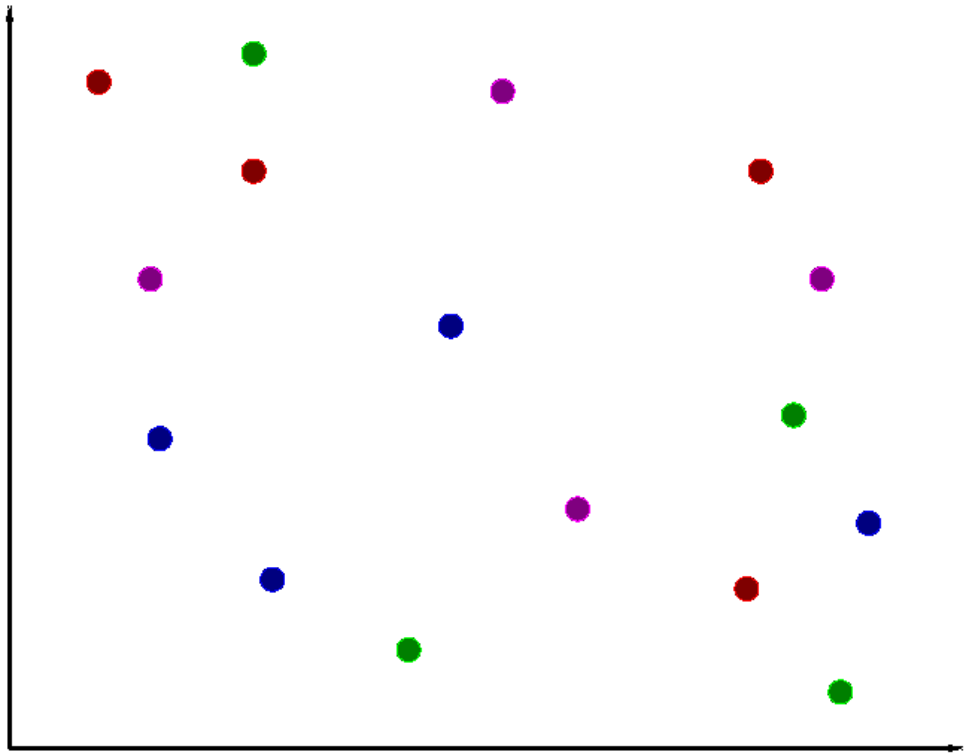
Basic Foundations of Particle Swarm Optimization

local best (lbest) PSO (cont)

Consider overlapping neighbourhoods based on spatial information (not all neighborhoods shown)



Based in particle indices

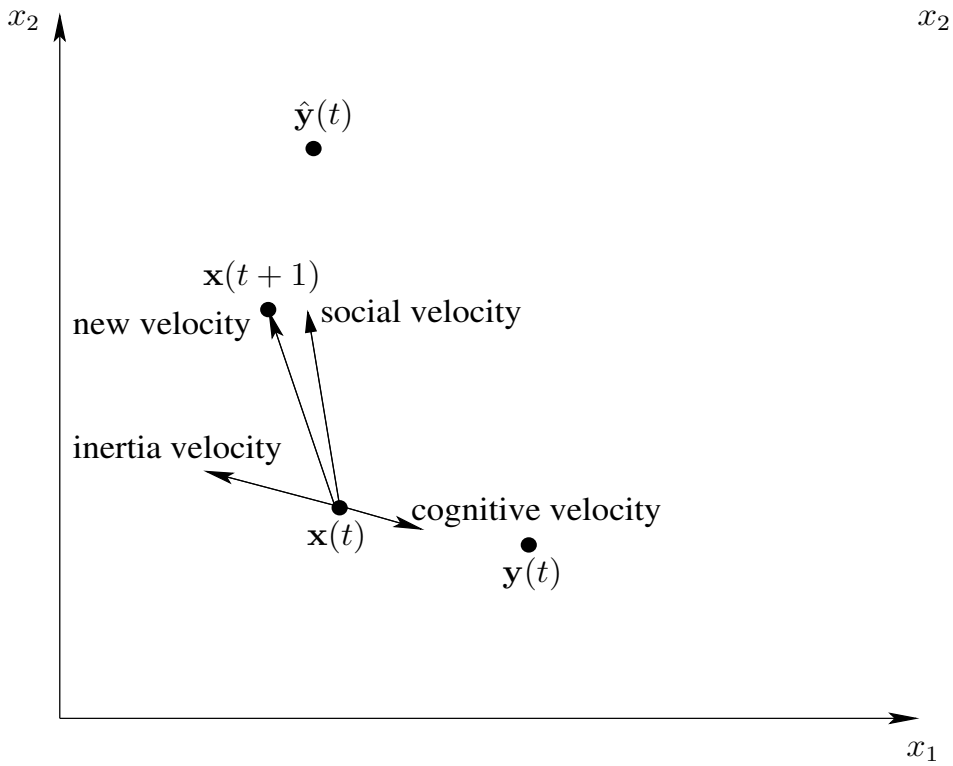


Create and initialize an n_x-dimensional swarm, S;
repeat
  for *each particle* $i = 1, \ldots, S.n_s$ **do**
    if $f(S.\mathbf{x}_i) < f(S.\mathbf{y}_i)$ **then**
      $S.\mathbf{y}_i = S.\mathbf{x}_i$;
    end
    for *each particle* $\hat{i}$ *with particle* i *in its neighborhood* **do**
      if $f(S.\mathbf{y}_i) < f(S.\hat{\mathbf{y}}_{\hat{i}})$ **then**
        $S.\hat{\mathbf{y}}_{\hat{i}} = S.\mathbf{y}_i$;
      end
    end
  end
  for *each particle* $i = 1, \ldots, S.n_s$ **do**
    update the velocity and position;
  end
until *stopping condition is true*;

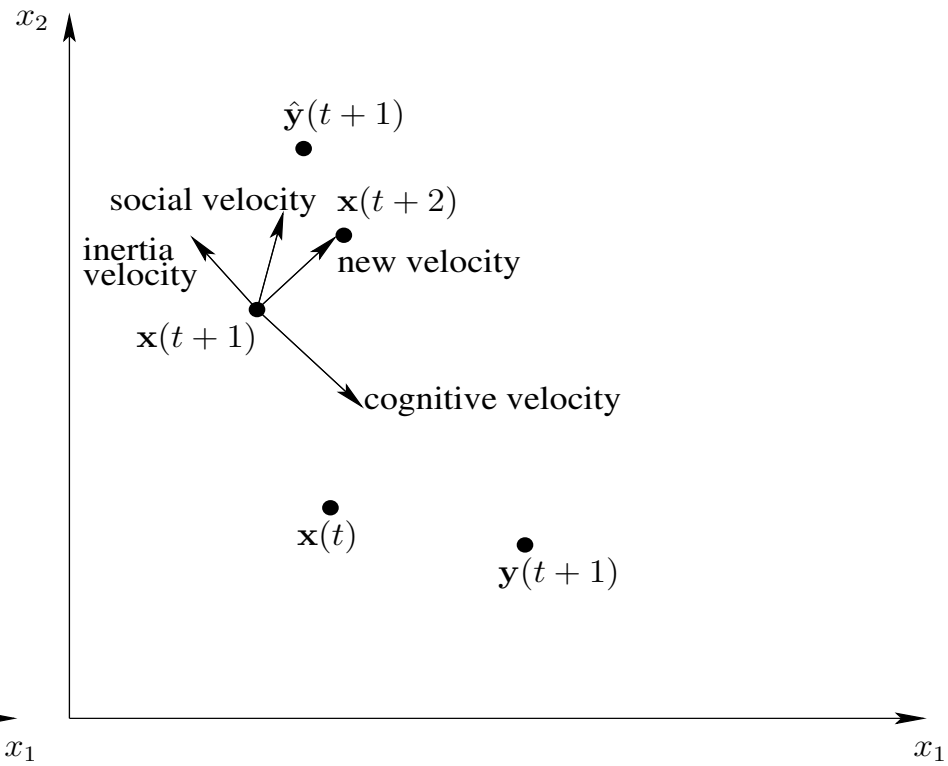
- previous velocity, $\mathbf{v}_i(t)$
  - inertia component
  - memory of previous flight direction
  - prevents particle from drastically changing direction
- cognitive component, $c_1\mathbf{r}_1(\mathbf{y}_i - \mathbf{x}_i)$
  - quantifies performance relative to past performances
  - memory of previous best position
  - nostalgia
- social component, $c_2\mathbf{r}_2(\hat{\mathbf{y}}_i - \mathbf{x}_i)$
  - quantifies performance relative to neighbors
  - envy

Basic Foundations of Particle Swarm Optimization

Geometric Illustration



(a) Time Step t



(b) Time Step $t+1$

- the problem: velocity quickly explodes to large values
- proposed solution:

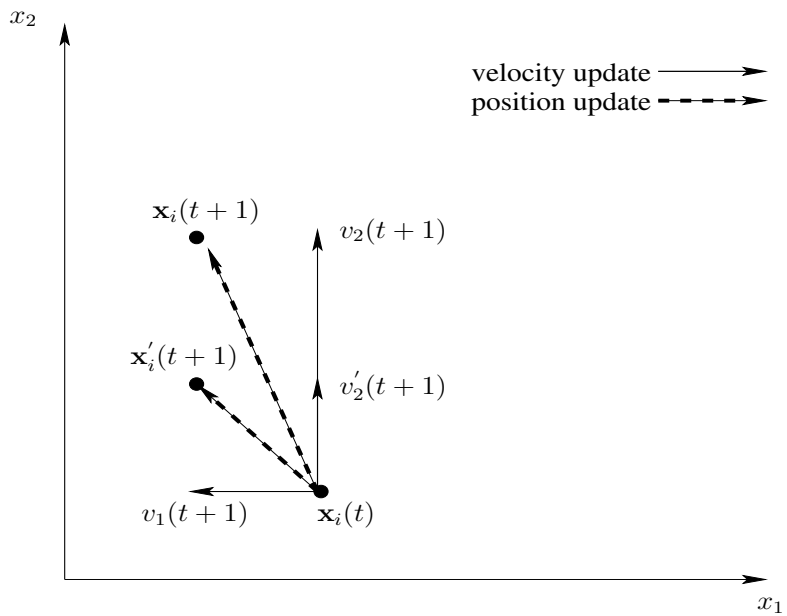
$$v_{ij}(t+1) = \begin{cases} v_{ij}(t+1) & \text{if } |v_{ij}(t+1)| < V_{max,j} \\ \text{sgn}(v_{ij}) V_{max,j} & \text{if } |v_{ij}(t+1)| \geq V_{max,j} \end{cases}$$

- controlling the global exploration of particles
- does not confine the positions, only the step sizes

Basic Foundations of Particle Swarm Optimization

Velocity Clamping (cont)

- Issues with velocity clamping:
  - dimensions with ranges smaller than V_{max} will never be clamped
  - if clamped in all directions, search will be in a straight line
  - changes search direction – normalized clamping



Change in Search Direction Due to Velocity Clamping

- problem-dependent
  - dynamically changing V_{max} when *gbest* does not improve over τ iterations

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

    β decreases from 1.0 to 0.01
  - exponentially decaying V_{max}

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

Basic Foundations of Particle Swarm Optimization

Inertia Weight

- to help in controlling exploration and exploitation, depending on the values of c_1 and c_2
- controls the momentum
- velocity update changes to

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

- for $w \geq 1$
  - velocities increase over time
  - swarm diverges
  - particles fail to change direction towards more promising regions
- for $0 < w < 1$
  - particles decelerate, depending on c_1 and c_2
- exploration–exploitation
  - large values – favor exploration
  - small values – promote exploitation
- problem-dependent

Dynamically changing inertia weights

- $w \sim N(0.72, \sigma)$
- linear decreasing

$$w(t) = (w(0) - w(n_t))\frac{(n_t - t)}{n_t} + w(n_t)$$

- non-linear decreasing

$$w(t+1) = \alpha w(t'), \quad w(0) = 1.4$$

- based on relative improvement

$$w_i(t+1) = w(0) + (w(n_t) - w(0))\frac{e^{m_i(t)} - 1}{e^{m_i(t)} + 1}$$

where the relative improvement, m_i, is estimated as

$$m_i(t) = \frac{f(\hat{\mathbf{y}}_i(t)) - f(\mathbf{x}_i(t))}{f(\hat{\mathbf{y}}_i(t)) + f(\mathbf{x}_i(t))}$$

- to ensure convergence to a stable point without the need for velocity clamping

$$\begin{aligned} v_{ij}(t+1) &= \chi[v_{ij}(t) + \phi_1(y_{ij}(t) - x_{ij}(t)) \\ &\quad + \phi_2(\hat{y}_j(t) - x_{ij}(t))] \end{aligned}$$

where

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

with

$$\begin{aligned} \phi &= \phi_1 + \phi_2 \\ \phi_1 &= c_1 r_1 \\ \phi_2 &= c_2 r_2 \end{aligned}$$

- if $\phi \geq 4$ and $\kappa \in [0, 1]$, then the swarm is guaranteed to converge to an equilibrium state
- $\chi \in [0, 1]$
- κ controls exploration–exploitation
  $\kappa \approx 0$: fast convergence, exploitation
  $\kappa \approx 1$: slow convergence, exploration
- effectively equivalent to inertia weight for specific χ:
  $w = \chi, \phi_1 = \chi c_1 r_1$ and $\phi_2 = \chi c_2 r_2$

- Synchronous iteration strategy
  - personal best and neighborhood bests updated separately from position and velocity vectors
  - slower feedback of new best positions
- Asynchronous iteration strategy
  - new best positions updated after each particle position update
  - immediate feedback of new best positions
  - lends itself well to parallel implementation
- Which strategy is best to use is very problem dependent

Synchronous Iteration Strategy

```
Create and initialize the swarm;
repeat
    for each particle do
        Evaluate particle's fitness;
        Update particle's personal
          best position;
        Update particle's
          neighborhood best
          position;
    end
    for each particle do
        Update particle's velocity;
        Update particle's position;
    end
until stopping condition is true;
```

Asynchronous Iteration Strategy

```
Create and initialize the swarm;
repeat
    for each particle do
        Update the particle's velocity;
        Update the particle's position;
        Evaluate particle's fitness;
        Update the particle's personal
          best position;
        Update the particle's
          neighborhood best position;
    end
until stopping condition is true;
```

- $c_1 = c_2 = 0$?
- $c_1 > 0, c_2 = 0$:
  - particles are independent hill-climbers
  - local search by each particle
  - cognitive-only PSO
- $c_1 = 0, c_2 > 0$:
  - swarm is one stochastic hill-climber
  - social-only PSO
- $c_1 = c_2 > 0$:
  - particles are attracted towards the average of $\mathbf{y}_i$ and $\hat{\mathbf{y}}_i$
- $c_2 > c_1$:
  - more beneficial for unimodal problems
- $c_1 > c_2$:
  - more beneficial for multimodal problems

- low c_1 and c_2:
  - smooth particle trajectories
- high c_1 and c_2:
  - more acceleration, abrupt movements
- problem dependent
  - adaptive acceleration coefficients, but note dependency on w

$$c_1(t) = (c_{1,min} - c_{1,max})\frac{t}{n_t} + c_{1,max}$$

$$c_2(t) = (c_{2,max} - c_{2,min})\frac{t}{n_t} + c_{2,min}$$

- Terminate when a maximum number of iterations, or FEs, has been exceeded
- Terminate when an acceptable solution has been found, i.e. when (assuming minimization)

$$f(\mathbf{x}_i) \leq |f(\mathbf{x}^*) - \epsilon|$$

- Terminate when no improvement is observed over a number of iterations

- Terminate when the normalized swarm radius is close to zero, i.e.

$$R_{norm} = \frac{R_{max}}{\text{diameter}(S(0))} \approx 0$$

where

$$R_{max} = ||\mathbf{x}_m - \hat{\mathbf{y}}||, \quad m = 1, \ldots, n_s$$

with

$$||\mathbf{x}_m - \hat{\mathbf{y}}|| \geq ||\mathbf{x}_i - \hat{\mathbf{y}}||, \quad \forall i = 1, \ldots, n_s$$

- Terminate when the objective function slope is approximately zero, i.e.

$$f'(t) = \frac{f(\hat{\mathbf{y}}(t)) - f(\hat{\mathbf{y}}(t-1))}{f(\hat{\mathbf{y}}(t))} \approx 0$$

Particles are guaranteed under certain conditions to converge to an equilibrium:

- In the limit, particles will converge to

$$\frac{c_1\mathbf{y} + c_2\hat{\mathbf{y}}}{c_1 + c_2}$$

- This is not necessarily even a local minimum
- It has been proven that standard PSO is not a local minimizer

Potential dangerous property:

- when $\mathbf{x}_i = \mathbf{y}_i = \hat{\mathbf{y}}_i$
- then the velocity update depends only on $w\mathbf{v}_i$
- if this condition persists for a number of iterations,

$$w\mathbf{v}_i \to 0$$

Simplified particle trajectories:

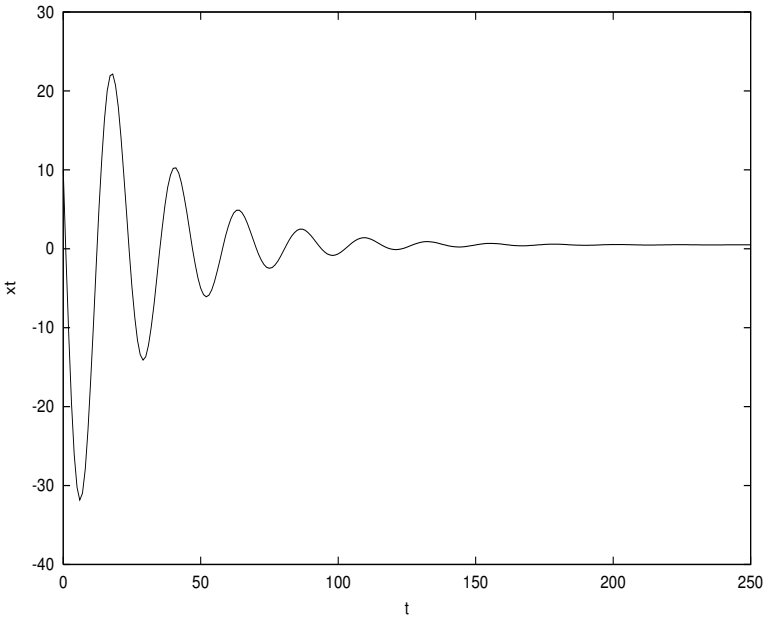
- no stochastic component
- single, one-dimensional particle
- using w
- personal best and global best are fixed: $y = 1.0, \hat{y} = 0.0$

Example trajectories:

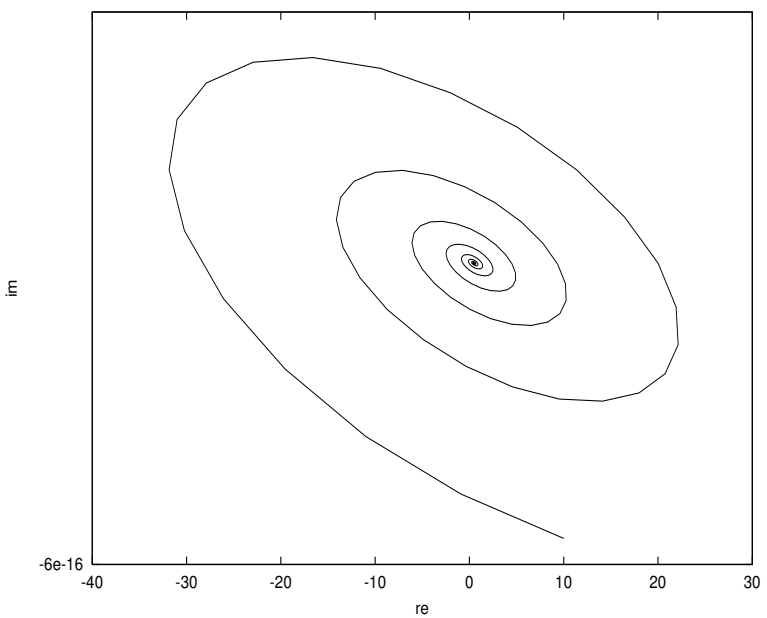
- Convergence to an equilibrium
- Cyclic behavior
- Divergent behavior

Convergence Behavior of Particle Swarm Optimization

Convergent Trajectories



(a) Time domain

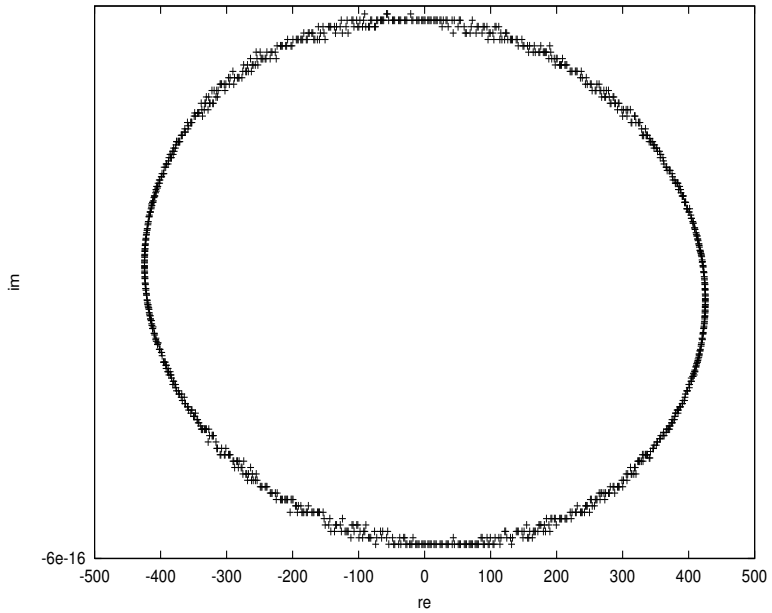


(b) Phase space

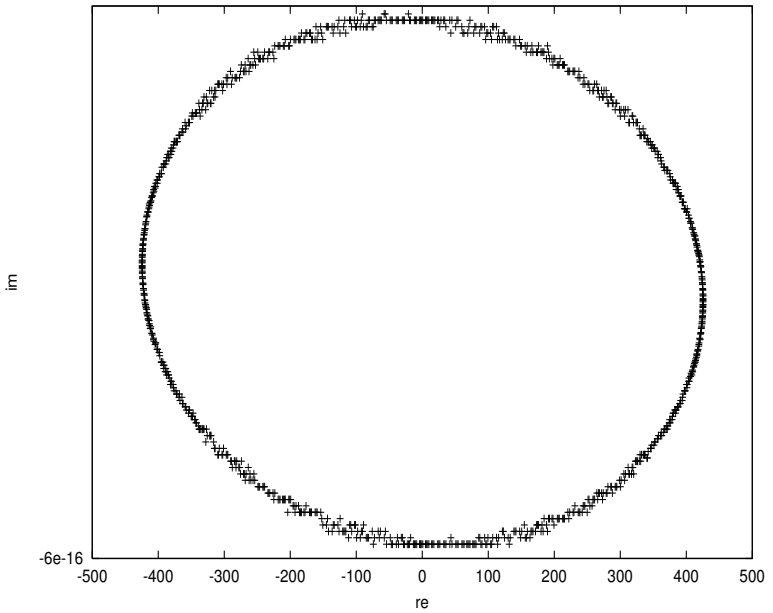
$w = 0.5$ and $\phi_1 = \phi_2 = 1.4$; $\phi_1 = c_1 r_1, \phi_2 = c_2 r_2$; $r_1 = r_2 = 1$

Convergence Behavior of Particle Swarm Optimization

Cyclic Trajectories



(a) Time domain

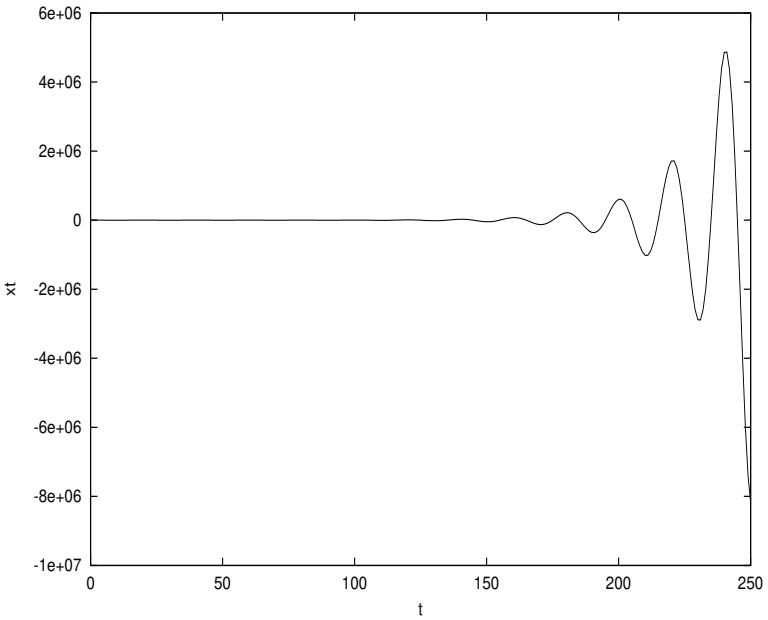


(b) Phase space

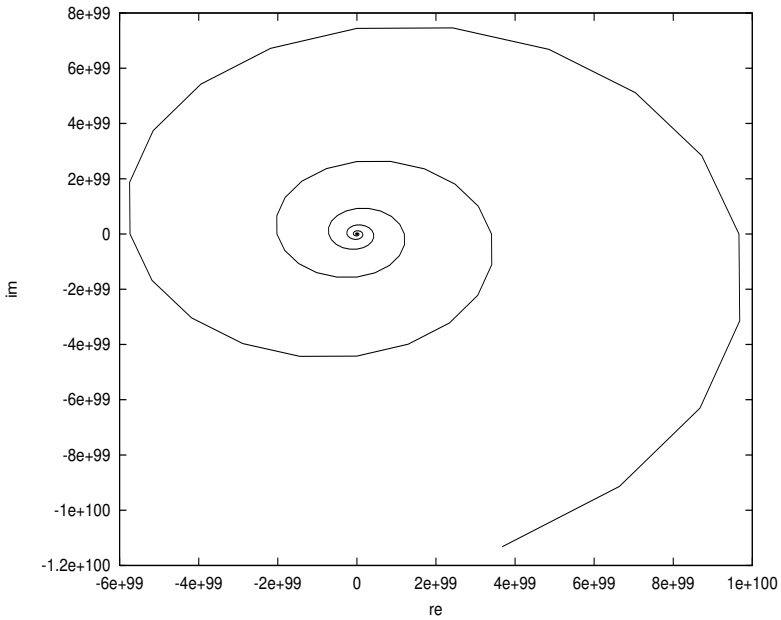
$$w = 1.0 \text{ and } \phi_1 = \phi_2 = 1.999$$

Convergence Behavior of Particle Swarm Optimization

Divergent Trajectories



(a) Time domain



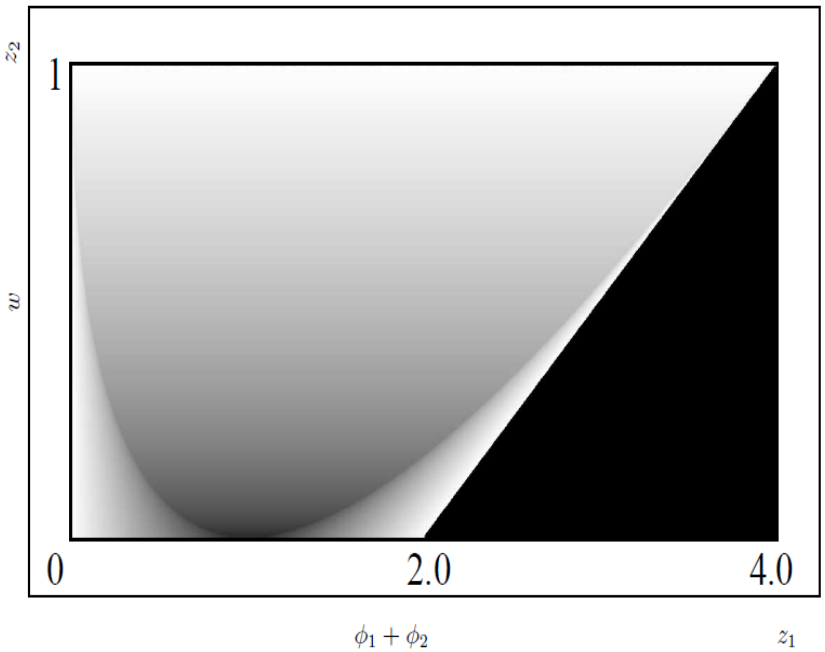
(b) Phase space

$w = 0.7$ and $\phi_1 = \phi_2 = 1.9$

Convergence Behavior of Particle Swarm Optimization

Stability Conditions

- What do we mean by the term convergence?
- Convergence map for values of w and $\phi = \phi_1 + \phi_2$, where $\phi_1 = c_1 r_1, \phi_2 = c_2 r_2$



Convergence Map for Values of w and $\phi = \phi_1 + \phi_2$

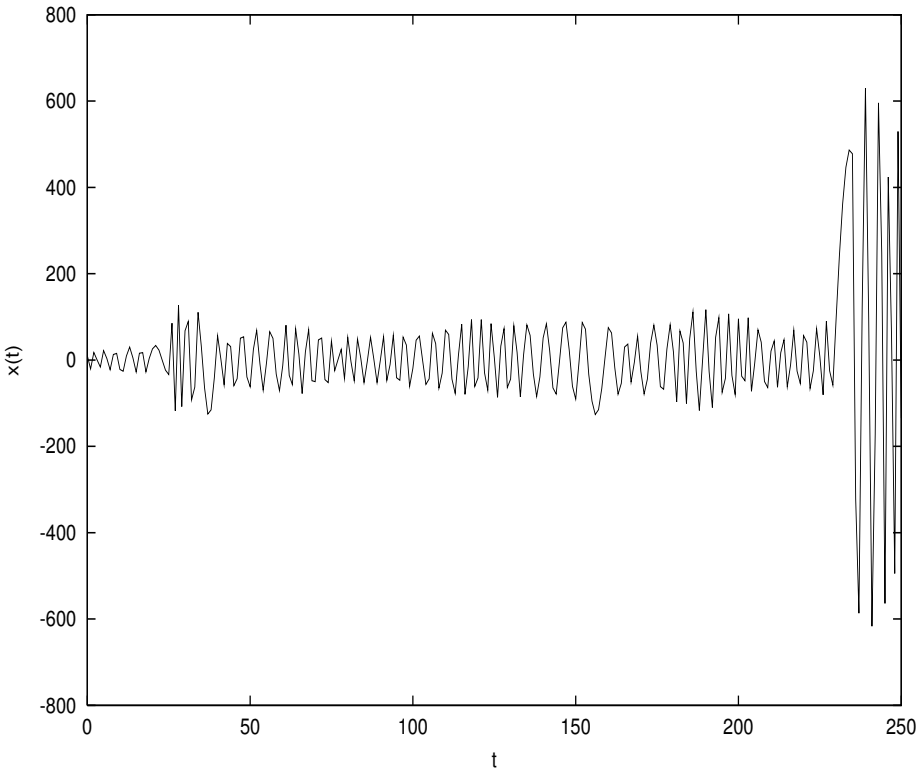
First stability condition on values of w, c_1 and c_2:

$$1 > w > \frac{1}{2}(\phi_1 + \phi_2) - 1 \geq 0$$

Convergence Behavior of Particle Swarm Optimization

Stochastic Trajectories

$w = 1.0$ and $c_1 = c_2 = 2.0$

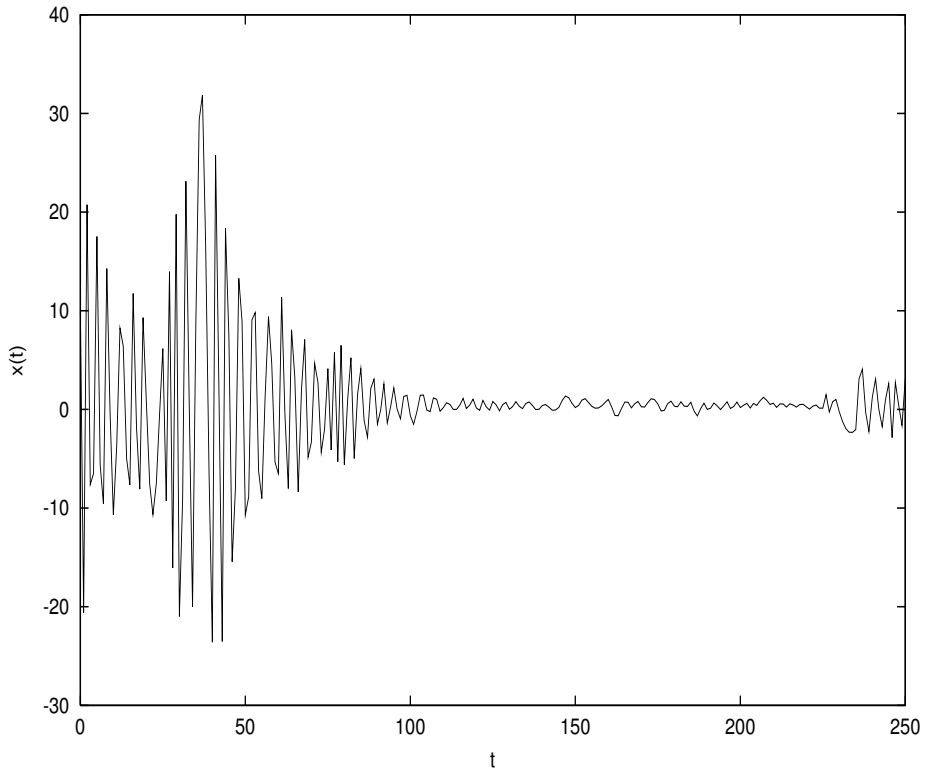


- violates the stability condition
- for $w = 1.0$, $c_1 + c_2 < 4.0$ to validate the condition

Convergence Behavior of Particle Swarm Optimization

Stochastic Trajectories (cont)

$w = 0.9$ and $c_1 = c_2 = 2.0$



- violates the stability condition
- for $w = 0.9$, $c_1 + c_2 < 3.8$ to validate the condition

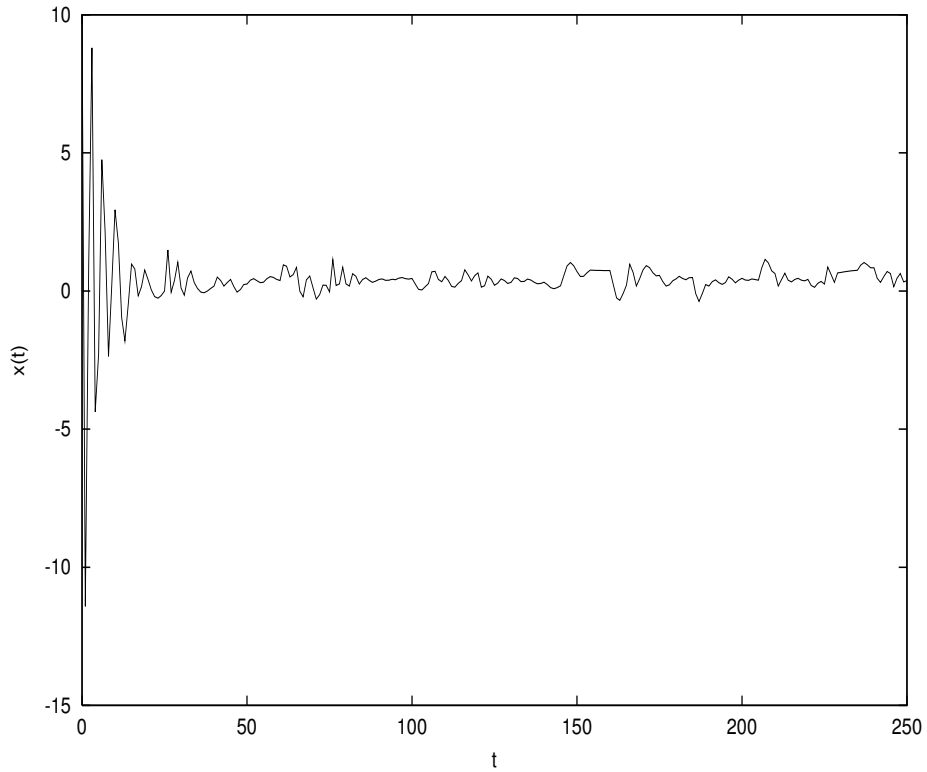
What is happening here?

- since $0 < \phi_1 + \phi_2 < 4$,
- and $r_1, r_2 \sim U(0, 1)$,
- prob$(\phi_1 + \phi_2 < 3.8) = \frac{3.8}{4} = 0.95$

Convergence Behavior of Particle Swarm Optimization

Stochastic Trajectories (cont)

$w = 0.7$ and $c_1 = c_2 = 1.4$



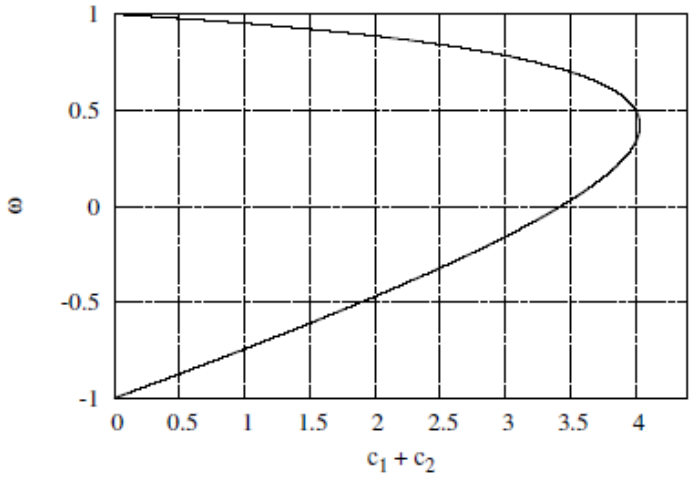
- validates the stability condition

Convergence Behavior of Particle Swarm Optimization

Convergence Conditions

A more recent and also the most accurate stability condition

$$c_1 + c_2 < \frac{24(1 - w^2)}{7 - 5w} \text{ for } w \in [-1, 1]$$



Empirically shown to be the best matching condition[1]

Note: Under unlimited computational budget

[1] CW Cleghorn, AP Engelbrecht, *Particle Swarm Convergence: An Empirical Investigation*, IEEE Congress on Evolutionary Computation 2014; CW Cleghorn, AP Engelbrecht, *Particle Swarm Variants: Standardized Convergence Analysis*, Swarm Intelligence, 9(2-3):177–203, 2015

Particle Roaming Behaviour[3]

The problem of particle roaming is a well known issue of PSO

- A particle is said to be roaming if it is moving outside of the defined problem boundaries, i.e. violates boundary constraints

Why do particles roam?

- It was formally proved by Helwig and Wanka[2] that particles will leave the search space with overwhelming probability in the first iteration
  - when velocities are uniformly initialized within $[-x_{min}, x_{max}]^{n_x}$ or initialized to $\mathbf{0}$.

---

[2] S Helwig and R Wanka, *Theoretical Analysis of Initial Particle Swarm Behavior*, in Proceedings of the Tenth International Conference on Parallel Problem Solving from Nature, 2008, pp. 889–898

[3] AP Engelbrecht, *Roaming Behavior of Unconstrained Particles*, BRICS-CCI 2013

Particle Roaming Behaviour

The Issues

- Empirical analysis and theoretical proofs showed that particles leave search boundaries very early during the optimization process
- Potential problems:
  - **Infeasible solutions**: Should better positions be found outside of boundaries, and no boundary constraint method employed, personal best and neighborhood best positions are pulled outside of search boundaries
  - **Wasted search effort**: Should better positions not exist outside of boundaries, particles are eventually pulled back into feasible space.
  - **Incorrect swarm diversity calculations**: As particles move outside of search boundaries, diversity increases

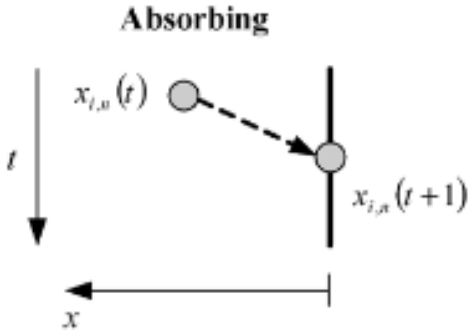
Some problems do not have boundary constraints, and then roaming is a good characteristic

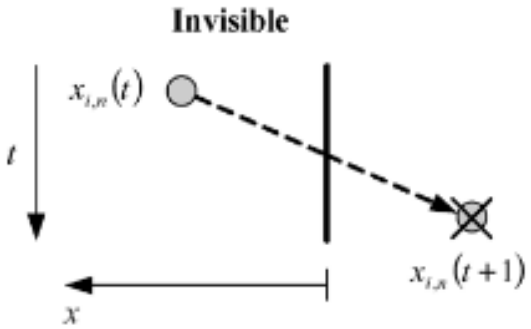
Particle Roaming Behaviour

Solutions

The roaming problem can be addressed by using boundary constraint mechanisms[4]

- Particle position repair approaches:
  - **Random**: re-initializes any invalid position component uniformly within the search space
  - **Absorb**: moves the particle back onto the boundary in every violated dimension, and velocity is set to zero
  - **Invisible**: Do not re-evaluate infeasible particle's fitness





[4] ET Oldewage, AP Engelbrecht, CW Cleghorn, *Boundary Constraint Handling Techniques for Particle Swarm Optimization in High Dimensional Problem Spaces*, Proceedings of the 11th International Swarm Intelligence Conference, Lecture Notes in Computer Science, volume 11172, pp 333–341, 2018

Particle Roaming Behaviour

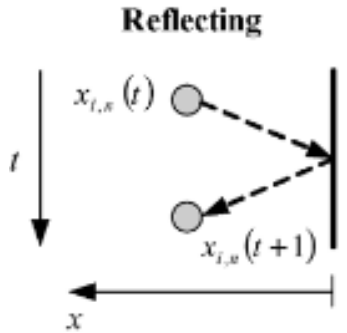
Solutions (cont)

- Particle position repair approaches:
  - **Exponential**: move violated components to position between particle's personal best position and violated boundary, using exponential distribution
  - **Infinity**: a simple approach is to update personal best positions only when solution quality improves AND the particle is feasible (within bounds)
    or
    set objective function value to worst possible value

Particle Roaming Behaviour

Solutions (cont)

- Velocity repair approaches:
  - **Adjust**: performs a position repair strategy, and then set velocity as $\mathbf{v}_i(t+1) = \mathbf{x}_i(t+1) - \mathbf{x}_i(t)$
  - **Reflect**: reflects the particle's velocity in the violated dimension, by reversing the sign of the velocity
  - **Damping**: boundary absorbs some of the velocity, and particle is reflected back with lesser velocity; fraction of velocity reduction is randomly decided



Particle Roaming Behaviour

Solutions (cont)

- Other strategies
  - **Pbest**: re-positions a particle to its personal best position and sets its velocity to 0 if it leaves the search space (in any dimension)
  - **Hyperbolic**: prevents a particle from ever reaching the boundary by scaling its velocity (the closer a particle is to the boundary, the smaller its scaled velocity is):

$$v_{i,j}^t = \begin{cases} \frac{v_{i,j}^t}{1+|v_{i,j}^t/(U_j - x_{i,j}^t)|} & \text{if } v_{ij}^t > \frac{U_j+L_j}{2} \\ \frac{v_{i,j}^t}{1+|v_{i,j}^t/(x_{i,j}^t - L_j)|} & \text{if } v_{ij}^t \leq \frac{U_j+L_j}{2} \end{cases}$$

The best approach to use is again very problem dependent

Velocity Initialization[5]

The Opinions

Velocties have been initialized using any of the following:

- $\mathbf{v}_i(0) = \mathbf{0}$
  - Critique: Limits exploration ability, therefore extent to which the search space is initially covered
  - Counter argument: Initial positions are uniformly distributed
  - Flocking analogy: Physical objects, in their initial state, do not have any momentum
- $\mathbf{v}_i(0) \sim U(-x_{min}, x_{max})^{n_x}$, where n_x is the problem dimension
  - Argument in favor: Initial random velocities help to improve exploration abilities of the swarm, therefore believed to obtain better solutions, faster
  - Argument against: large initial step sizes cause more particles to leave search boundaries and for longer:

$$\mathbf{v}_i(0) \sim U(-x_{min}, x_{max})^{n_x} \longrightarrow \mathbf{x}_i(1) \sim U(-2x_{min}, 2x_{max})^{n_x}$$

  - More roaming!

[5] AP Engelbrecht, *Particle Swarm Optimization: Velocity Initialization*, IEEE Congress on Evolutionary Computation 2012

A rough classification of PSO algorithms:

- Social-based algorithms
  - Different socal network structures
  - Different information sharing strategies
- Hybrid algorithms
- Sub-swarm-based algorithms
- Memetic algorithms
- Multi-start algorithms
- Heterogeneous algorithms
- Self-adaptive algorithms

Social-based algorithms are algorithms that use different

- neighborhood topologies, and/or
- information sharing approaches

Example algorithms

- gbest PSO versus lbest PSO versus Von Neumann PSO
- spatial neighborhoods PSO
- fitness-based spatial neighborhoods PSO
- growing neighborhoods PSO
- fully informed PSO
- barebones PSO

Inertia PSO:

- Velocity update:

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

  where $\hat{\mathbf{y}}_i$ is the neighborhood best

- Position update

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

Using spatial neighborhoods:

Calculate the Euclidean distance $\mathcal{E}(\mathbf{x}_{i_1}, \mathbf{x}_{i_2}), \forall i_1, i_2 = 1, \ldots, n_s$;
$S = \{i : i = 1, \ldots, n_s\}$;
for $i = 1, \ldots, n_s$ **do**
    $S' = S$;
    for $i' = 1, \ldots, n_{\mathcal{N}_{i'}}$ **do**
        $\mathcal{N}_i = \mathcal{N}_i \cup \{\mathbf{x}_{i''} : \mathcal{E}(\mathbf{x}_i, \mathbf{x}_{i''}) = \min\{\mathcal{E}(\mathbf{x}_i, \mathbf{x}_{i'''}), \forall \mathbf{x}_{i'''} \in S'\}$;
        $S' = S' \setminus \{\mathbf{x}_{i''}\}$;
    end
end

May result in disconnected neighborhoods, which

- limits exploration, and
- result in convergence to different points in the search space

Improvements to spatial neighborhoods:

- penalize bad neighbours
  - add penalty to distance proportional to quality of neighbour's solution
- Growing neighborhoods:
  - start with the ring topology
  - linearly increase the number of neighbours
  - end with the star topology

Fully Informed PSO:

- Velocity equation is changed such that each particle is influenced by the successes of all its neighbors, and not on the performance of only one individual
- Each particle in the neighborhood, $\mathcal{N}_i$, of particle i is regarded equally:

$$\mathbf{x}_i(t+1) = \chi\left(\mathbf{v}_i(t) + \sum_{m=1}^{n_{\mathcal{N}_i}} \frac{\mathbf{r}(t)(\mathbf{y}_m(t) - \mathbf{x}_i(t))}{n_{\mathcal{N}_i}}\right)$$

where $n_{\mathcal{N}_i} = |\mathcal{N}_i|$, and $\mathbf{r}(t) \sim U(0, c_1 + c_2)^{n_x}$

- A weight is assigned to the contribution of each particle based on the performance of that particle:

$$\mathbf{x}_i(t+1) = \chi \left(\mathbf{v}_i(t) + \frac{\sum_{m=1}^{n_{\mathcal{N}_i}} \left(\frac{\phi_m \mathbf{p}_m(t)}{f(\mathbf{x}_m(t))} \right)}{\sum_{m=1}^{n_{\mathcal{N}_i}} \left(\frac{\phi_m}{f(\mathbf{x}_m(t))} \right)} \right)$$

where $\phi_m \sim U(0, \frac{c_1+c_2}{n_{\mathcal{N}_i}})$, and

$$\mathbf{p}_m(t) = \frac{\phi_1 \mathbf{y}_m(t) + \phi_2 \hat{\mathbf{y}}_m(t)}{\phi_1 + \phi_2}$$

Barebones PSO:

- Formal proofs have shown that each particle converges to a point that is a weighted average between the personal best and neighborhood best positions:

$$\frac{y_{ij}(t) + \hat{y}_{ij}(t)}{2}$$

- This behavior supports Kennedy's proposal to replace the entire velocity by

$$v_{ij}(t+1) \sim N\left(\frac{y_{ij}(t) + \hat{y}_{ij}(t)}{2}, \sigma\right)$$

where

$$\sigma = |y_{ij}(t) - \hat{y}_{ij}(t)|$$

- The position update is simply

$$x_{ij}(t+1) = v_{ij}(t+1)$$

- Alternative formulation:

$$v_{ij}(t+1) = \begin{cases} y_{ij}(t) & \text{if } U(0,1) < 0.5 \\ N(\frac{y_{ij}(t)+\hat{y}_{ij}(t)}{2}, \sigma) & \text{otherwise} \end{cases}$$

- There is a 50% chance that the j-th dimension of the particle dimension changes to the corresponding personal best position

Hybrid algorithms combine PSO with components of algorithms from other paradigms such as evolutionary algorithms:

- Selection-based PSO
- Reproduction in PSO
- Mutation in PSO
- Differential evolution based PSO

Selection-Based PSO:

Calculate the fitness of all particles;
for *each particle* $i = 1, \ldots, n_s$ **do**
    Randomly select n_{ts} particles;
    Score the performance of particle i against the n_{ts} randomly selected particles;
end
Sort the swarm based on performance scores;
Replace the worst half of the swarm with the top half, without changing the personal best positions;

- Any problems with this?
- Any advantages?

Using arithmetic crossover:

- An arithmetic crossover operator to produce offspring from two randomly selected particles:

$$\begin{aligned}\mathbf{x}_{i_1}(t+1) &= \mathbf{r}(t)\mathbf{x}_{i_1}(t) + (\mathbf{1} - \mathbf{r}(t))\mathbf{x}_{i_2}(t)\\ \mathbf{x}_{i_2}(t+1) &= \mathbf{r}(t)\mathbf{x}_{i_2}(t) + (\mathbf{1} - \mathbf{r}(t))\mathbf{x}_{i_1}(t)\end{aligned}$$

with the corresponding velocities,

$$\begin{aligned}\mathbf{v}_{i_1}(t+1) &= \frac{\mathbf{v}_{i_1}(t) + \mathbf{v}_{i_2}(t)}{||\mathbf{v}_{i_1}(t) + \mathbf{v}_{i_2}(t)||}||\mathbf{v}_{i_1}(t)||\\ \mathbf{v}_{i_2}(t+1) &= \frac{\mathbf{v}_{i_1}(t) + \mathbf{v}_{i_2}(t)}{||\mathbf{v}_{i_1}(t) + \mathbf{v}_{i_2}(t)||}||\mathbf{v}_{i_2}(t)||\end{aligned}$$

where $\mathbf{r}_1(t) \sim U(0,1)^{n_x}$

- Personal best position of an offspring is initialized to its current position:

$$\mathbf{y}_{i_1}(t+1) = \mathbf{x}_{i_1}(t+1)$$

- Particles are selected for breeding at a user-specified breeding probability
- Random selection of parents prevents the best particles from dominating the breeding process
- Breeding process is done for each iteration after the velocity and position updates have been done

- Disadvantage:
  - Parent particles are replaced even if the offspring is worse off in fitness
  - If $f(\mathbf{x}_{i_1}(t+1)) > f(\mathbf{x}_{i_1}(t))$ (assuming a minimization problem), replacement of the personal best with $\mathbf{x}_{i_1}(t+1)$ loses important information about previous personal best positions
- Solutions:
  - Replace parent with its offspring only if fitness of offspring is better than that of the parent.
  - Or, use Boltzmann selection

[6] See AP Engelbrecht, *Particle Swarm Optimization with Crossover: A Review and Empirical Analysis*, Artificial Intelligence Review, 45(2):131–165, 2016

Mutation PSO:

- Mutate the global best position:

$$\hat{\mathbf{y}}(t+1) = \hat{\mathbf{y}}'(t+1) + \eta' \mathbf{N}(0,1)$$

  where $\hat{\mathbf{y}}'(t+1)$ represents the unmutated global best position, and η' is referred to as a learning parameter

- Mutate the components of position vectors: For each j, if $U(0,1) < P_m$, then component $x'_{ij}(t+1)$ is mutated using

$$x_{ij}(t+1) = x'_{ij}(t+1) + N(0,\sigma)x'_{ij}(t+1)$$

  where

$$\sigma = 0.1(x_{max,j} - x_{min,j})$$

- Each x_{ij} can have its own deviation:

$$\sigma_{ij}(t) = \sigma_{ij}(t-1)\, e^{\tau' N(0,1) + \tau N_j(0,1)}$$

with

$$\tau' = \frac{1}{\sqrt{2\sqrt{n_x}}}$$

$$\tau = \frac{1}{\sqrt{2n_x}}$$

Sub-swarm-based algorithms utilize multiple swarms and/or sub-swarms:

- Cooperative Split PSO (see discussion on large scale optimization)
- Hybrid Cooperative Split PSO (see large scale optimization section)
- Predator-Prey PSO
- Life-cycle PSO
- Attractive and Repulsive PSO
- Vector-evaluated PSO (see multi-objective optimization section)
- NichePSO (see multi-modal optimization section)
- Multi-swarm PSO

Predator-Prey PSO:

- Competition introduced to balance exploration–exploitation
- Uses a second swarm of predator particles:
  - Prey particles scatter (explore) by being repelled by the presence of predator particles
  - Silva *et al.*[7] use only one predator
  - The velocity update for the predator particle is defined as

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

    where $\mathbf{v}_p$ and $\mathbf{x}_p$ are respectively the velocity and position vectors of the predator particle, p
    $\mathbf{r} \sim U(0, V_{max,p})^{n_x}$
  - $V_{max,p}$ controls the speed at which predator catches best prey

[7] A Silva, A Neves, E Costa, *An Empirical Comparison of Particle Swarm and Predator Prey Optimisation*, Proceedings of the Thirteenth Irish Conference on Artificial Intelligence and Cognitive Science, Lecture Notes in Artificial Intelligence", 2464:103–110, 2002

- The prey particles update their velocity using

$$\begin{aligned} v_{ij}(t+1) &= wv_{ij}(t) + c_1 r_{1j}(t)(y_{ij}(t) - x_{ij}(t)) \\ &+ c_2 r_{2j}(t)(\hat{y}_j(t) - x_{ij}(t)) \\ &+ c_3 r_{3j}(t) D(d) \end{aligned}$$

  d is Euclidean distance between prey, i, and the predator
  $r_{3j}(t) \sim U(0,1)$, and $D(d) = \alpha\, e^{-\beta d}$
- $D(d)$ quantifies the influence that the predator has on the prey, growing exponentially with proximity
- Position update of prey particles: If $U(0,1) < P_f$, then the prey velocity update above is used, otherwise the normal velocity update is used

Life-Cycle PSO:

Initialize a population of individuals
repeat
  for *all individuals* **do**
    Evaluate fitness;
    if *fitness did not improve* **then**
      Switch to next phase;
    end
  end
  for *all PSO particles* **do**
    Calculate new velocity vectors;
    Update positions;
  end
until *stopping condition is true*;

for *all GA individuals* **do**
  Perform reproduction;
  Mutate;
  Select new population;
end
for *all Hill-climbers* **do**
  Find possible new neighboring solution;
  Evaluate fitness of new solution;
  Move to new solution with specified probability;
end

Memetic PSO algorithms incorporate some form of local search:

- Guaranteed Convergence PSO
- PSO with Gradient Descent

Guaranteed Convergence PSO:

- Forces the global best position to change when

$$\mathbf{x}_i = \mathbf{y}_i = \hat{\mathbf{y}}$$

- The global best particle is forced to search in a bounding box around the current position for a better position
- The position update of the global best particle changes to

$$x_{\tau j}(t+1) = \hat{y}_j(t) + wv_{\tau j}(t) + \rho(t)(1 - 2r_2(t))$$

  where τ is the index of the global best particle
- The velocity update of the global best particle changes to

$$v_{\tau j}(t+1) = -x_{\tau j}(t) + \hat{y}_j(t) + wv_{\tau j}(t) + \rho(t)(1 - 2r_{2j}(t))$$

  where where $\rho(t)$ is a scaling factor

[8] F van den Bergh, AP Engelbrecht, *A New Locally Convergent Particle Swarm Optimiser*, IEEE International Conference on Systems, Man, and Cybernetics, 2002

- The scaling factor is defined as

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

  where $\#successes$ and $\#failures$ respectively denote the number of consecutive successes and failures
- A failure is defined as $f(\hat{\mathbf{y}}(t)) \leq f(\hat{\mathbf{y}}(t+1))$
- Proven to converge to at least a local minimum[9]

[9] F van den Bergh, AP Engelbrecht, *A Convergence Proof for the Particle Swarm Optimizer*, Fundamenta Informaticae, 105(4):341-374, 2010

Gradient-based PSO:

- Each particle computes two positions in the same direction:

$$\mathbf{x}_i^{'}(t+1) = \mathbf{x}_i(t) + \mathbf{v}_i(t+1)$$

$$\mathbf{x}_i^{''}(t+1) = \mathbf{x}_i(t) + \beta\mathbf{v}_i(t+1)$$

  where $\beta \in (0,1)$.
- The position in each dimension is then determined as

$$x_{ij}(t+1) = \arg\max \left\{ -\frac{f(x_{ij}^{''}(t+1)) - f(x_{ij}(t))}{\beta}, -\frac{f(x_{ij}^{'}(t+1)) - f(x_{ij}(t))}{\beta} \right\}$$

- Alternative: gbest position moves in the negative gradient

- Major problems with the basic PSO is lack of diversity when particles start to converge to the same point
- Multi-start methods have as their main objective to increase diversity, whereby larger parts of the search space are explored
- This is done by continually injecting randomness, or chaos, into the swarm
- Note that continual injection of random positions will cause the swarm never to reach an equilibrium state
- Methods should reduce chaos over time to ensure convergence

Craziness PSO:

- *Craziness* is the process of randomly initializing some particles
- What are the issues in reinitialization?
  - What should be randomized?
  - When should randomization occur?
  - How should it be done?
  - Which members of the swarm will be affected?
  - What should be done with personal best positions of affected particles?

- PSO performance is very sensitive to control parameter values
- Approaches to find the best values for control parameters:
  - Just use the values published in literature?
    - Do you want something that works – always,
    - or something that works best?
  - Fine-tuned static values
  - Dynamically changing values
  - Self-adaptive control parameters
  - Many dynamic and self-adaptive approaches have recently been developed, but... more research is needed...

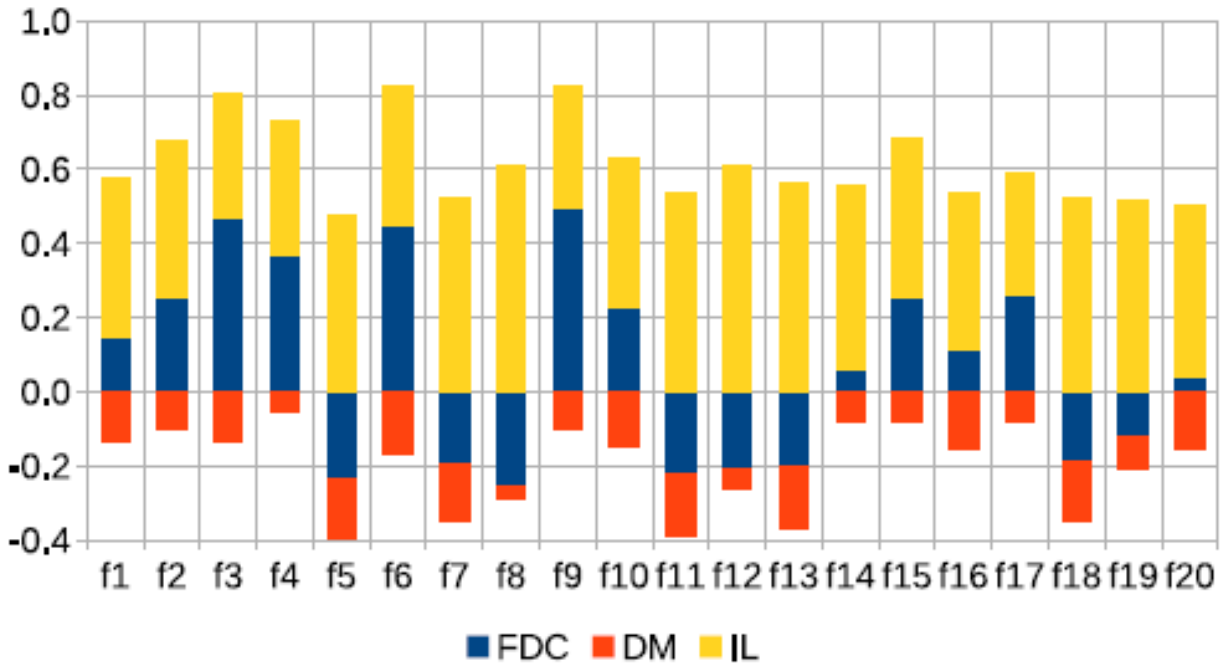
Self-Adaptive Particle Swarm Optimization

Introduction (cont)

To illustrate the dependence of optimal values of the control parameters on the problem being solved:

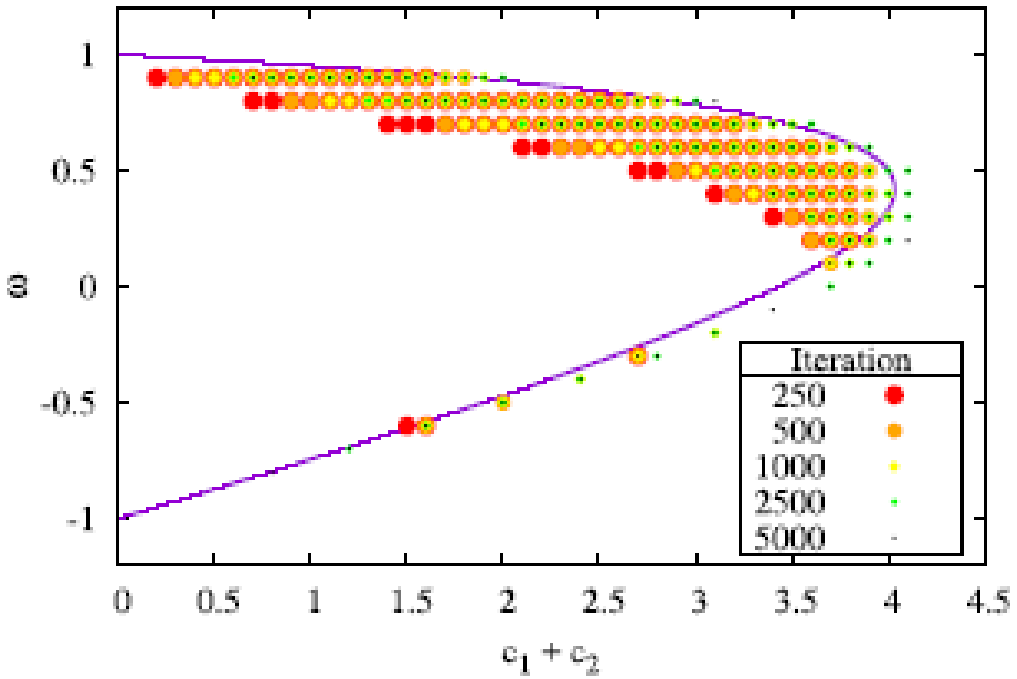
Function	w, c_1, c_2	Function	w, c_1, c_2	Function	w, c_1, c_2
Ackley 1	0.6, 2.0, 1.8	Levy 3	0.8, 1.8, 1.0	Rastrigin	0.85, 1.4, 0.4
Alpine 1	0.65, 1.6, 1.8	LevyMontalvo	0.8, 2.0, 0.8	Ripple 25	0.8, 0.8, 1.8
Bohachevsky 1	0.75, 1.2, 2.0	Mishra 1	0.75, 2.0, 1.0	Rosenbrock	0.85, 1.2, 1.0
BMF	0.55, 1.8, 1.8	Mishra 4	0.85, 1.2, 0.8	Salomon	0.75, 1.6, 1.4
Brown	0.6, 1.8, 1.6	Mishra 7	0.8, 0.2, 1.6	Schwefel 1	0.5, 2.0, 1.8
CosineMixture	0.8, 0.2, 1.8	NeedleEye	1.0, 0.2, 0.2	Schwefel 2.26	0.8, 0.2, 2.0
CrossInTray	0.75, 2.0, 0.4	Norwegian	0.65, 0.6, 2.0	Shubert 4	0.8, 0.2, 2.0
Discus	0.75, 2.0, 1.0	Paviani	0.6, 1.4, 1.8	Step 3	0.15, 2.0, 2.0
DropWave	0.8, 2.0, 0.6	Penalty 1	0.4, 2.0, 2.0	Trigonometric	0.8, 1.8, 0.6
Easom	0.8, 1.0, 0.4	Penalty 2	0.7, 1.8, 1.5	Vincent	0.55, 1.8, 1.8
Elliptic	0.5, 1.6, 2.0	Pinter 2	0.8, 2.0, 0.8	Weierstrass	0.9, 2.0, 1.0
EggCrate	0.85, 1.8, 0.2	Price 2	0.65, 1.2, 1.8	XinSheYang 1	0.85, 1.6, 0.6
EggHolder	0.8, 0.2, 2.0	Qings	0.5, 1.6, 2.0	XinSheYang 3	0.75, 0.2, 2.0
Exponential	0.45, 1.8, 2.0	Quadric	0.6, 1.4, 1.8	XinSheYang 4	0.6, 1.0, 2.0
Giunta	0.8, 2.0, 0.8	Rana	0.8, 0.2, 2.0		

Fitness landscape analysis for 10-dimensional problems



FDC: Fitness distance correlation (quantifies the correlation between the fitness of a solution and its distance to the nearest optimum)
DM: Dispersion metric (indication of funnels)
IL: Information landscape (searchability of the landscape)

Optimality of control parameter values change over time



Approaches to the control parameter problem:

- Factorial design
- F-Race
- Control parameter dependencies
- Multiple performance objectives
- Problem dependency
- Computationally expensive

Control Parameters[10]

Self-Adaptive Particle Swarm Optimization: Shortcomings

Issues with current self-adaptive approaches:

- Most, at some point in time, violate convergence conditions, and many do so for most of the search process
- Converge prematurely, with little exploration of control parameter space
- Introduce more control parameters
- Current empirical analysis shows that they do not really result in improved performance with reference to solution quality

[10] KR Harrison, AP Engekbrecht, BM Ombuki-Berman, *Self-Adaptive Particle Swarm Optimization: A Review and Analysis of Convergence*, Swarm Intelligence, 12:187–226, 2018

Control Parameters

Self-Adaptive Particle Swarm Optimization: Approaches

Optimizer	Parameters Tuned	Net Change
PSO-TVIW (Shi and Eberhart, 1998, 1999)	ω	+1
PSO-AIWF (Liu et al, 2005)	ω	+1
DAPSO (Yang et al, 2007)	ω	+2
IPSO-LT (Li and Tan, 2008)	ω	+1
SAPSO-LFZ (Li et al, 2008)	ω	-1 (0)
SAPSO-DWCY (Dong et al, 2008)	ω	-1 (+2)
PSO-RBI (Panigrahi et al, 2008)	ω	+1
IPSO-CLL (Chen et al, 2009)	ω	-1
AIWPSO (Nickabadi et al, 2011)	ω	+1
APSO-VI (Xu, 2013)	ω	+2
SRPSO (Tanweer et al, 2015)	ω	+2
PSO-SAIC (Wu and Zhou, 2007)	ω, c_2	+2 (+4)
PSO-RAC	ω, c_1, c_2	-3
PSO-TVAC (Ratnaweera et al, 2004)	ω, c_1, c_2	+3
PSO-ICSA (Jun and Jian, 2009)	ω, c_1, c_2	+3 (+31)
APSO-ZZLC (Zhan et al, 2009)	ω, c_1, c_2	-3 (+35)
UAPSO-A (Hashemi and Meybodi, 2011)	ω, c_1, c_2	+6
GPSO (Leu and Yeh, 2012)	ω, c_1, c_2	+3 (+(n_d + 3))

- Time-Varying Inertia Weight (PSO-TVIW)

$$w(t) = w_s + (w_f - w_s)\frac{t}{n_t}$$

where w_s and w_f are the initial and final inertia weight values, n_t is the maximum number of iterations

- Time-Varying Acceleration Coefficients (PSO-TVAC)

$$c_1(t) = c_{1s} + (c_{1f} - c_{1s})\frac{t}{n_t}$$

$$c_2(t) = c_{2s} + (c_{2f} - c_{2s})\frac{t}{n_t}$$

PSO with Simulated Annealing:

- Inertia weight is adapted as

$$w_i(t) = w_a F(\eta_i(t)) + w_b$$

where w_a and w_b are user-specified positive constants, and

$$F(\eta_i(t)) = \begin{cases} 2 & \text{if } \eta_i(t) < 0.0001 \\ 1 & \text{if } 0.0001 \leq \eta_i(t) < 0.01 \\ 0.3 & \text{if } 0.01 \leq \eta_i(t) < 0.1 \\ -0.8 & \text{if } 0.1 \leq \eta_i(t) < 0.9 \\ -5.5 & \text{if } 0.9 \leq \eta_i(t) \leq 1.0 \end{cases}$$

and the relative particle performance is

$$\eta_i(t) = \frac{f(\hat{\mathbf{y}}_i(t-1))}{f(\mathbf{x}_i(t-1))}$$

$\eta_i(t) \approx 0$ denotes that particle is much worse than the nbest
$\eta_i(t) = 1$ denotes particle is as good as nbest

- Social acceleration adapted as

$$c_{2i}(t) = c_{2a}G(\eta_i(t)) + c_{2b}$$

and

$$G(\eta_i(t)) = \begin{cases} 2.5 & \text{if } \eta_i(t) < 0.0001 \\ 1.2 & \text{if } 0.0001 \leq \eta_i(t) < 0.01 \\ 0.5 & \text{if } 0.01 \leq \eta_i(t) < 0.1 \\ 0.2 & \text{if } 0.1 \leq \eta_i(t) < 0.9 \\ 0.1 & \text{if } 0.9 \leq \eta_i(t) \leq 1.0 \end{cases}$$

For η_i low, c_2 increases

Improved Particle Swarm Optimization adapts inertia weight as:

$$w(t) = e^{-\lambda(t)}$$

with

$$\lambda(t) = \frac{\alpha(t)}{\alpha(t-1)}$$

and

$$\alpha(t) = \frac{1}{n_s} \sum_{i=1}^{n_s} |f(\mathbf{x}_i(t)) - f(\hat{\mathbf{y}}(t))|$$

Self-Adaptive Particle Swarm Optimization

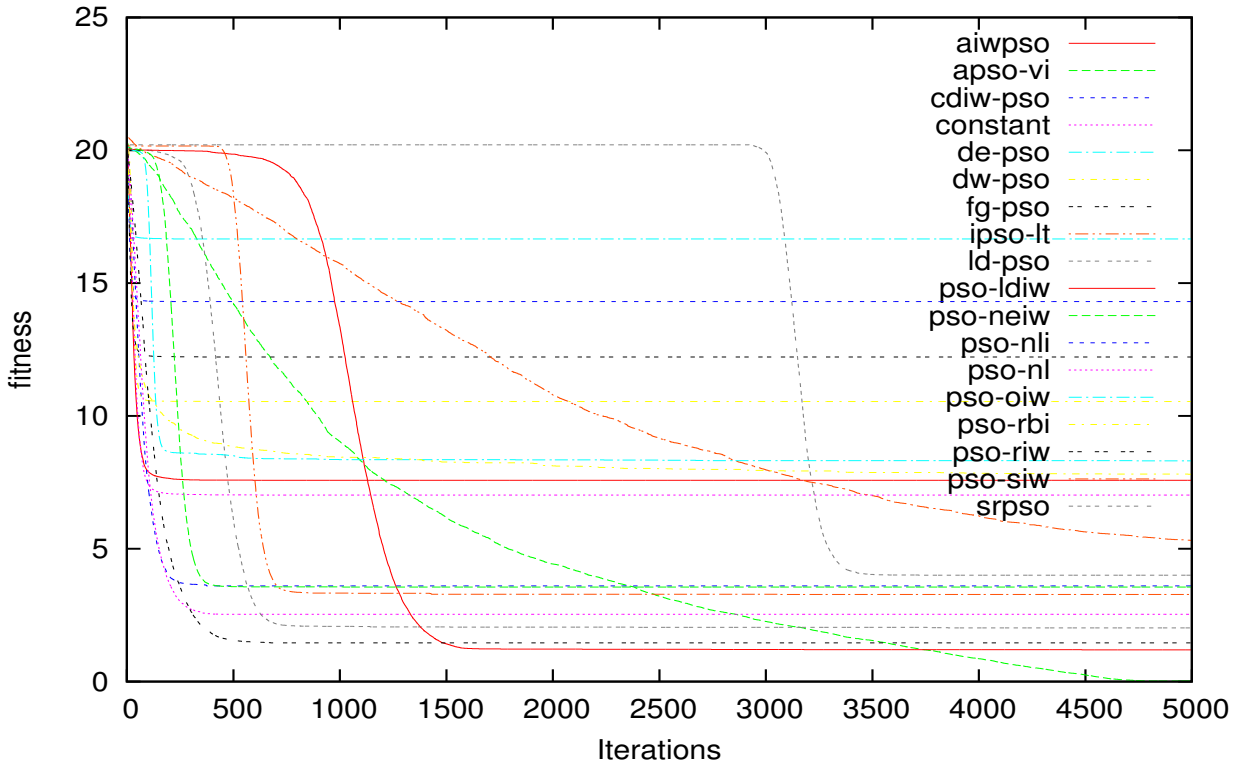
Self-Adaptive Control Parameters (cont)

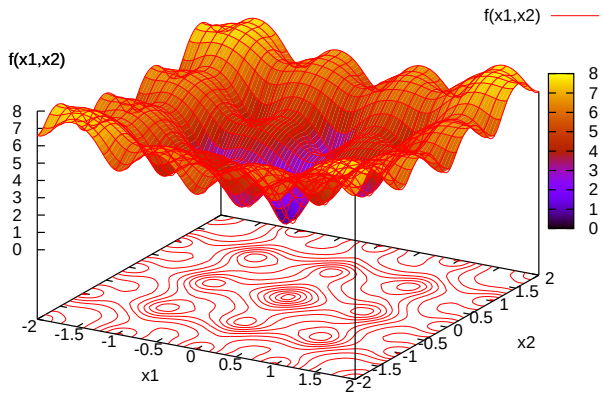
The Self-Adaptive PSO, adapts inertia weight as

$$w_i(t) = 0.15 + \frac{1}{1 + e^{\overline{f(\mathbf{y}(t))} - f(\mathbf{y}_i(t))}}$$

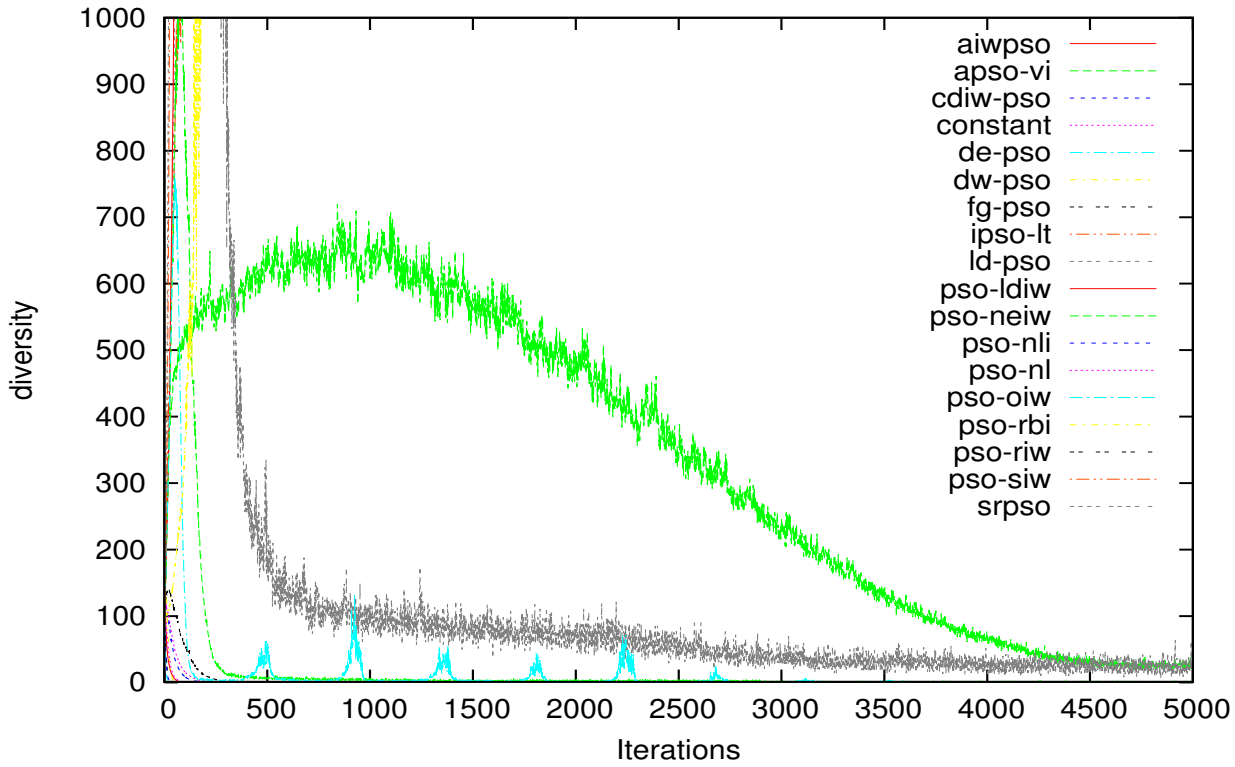
where $\overline{f(\mathbf{y}(t))}$ is the average pbest fitness values of the swarm

Average solution quality

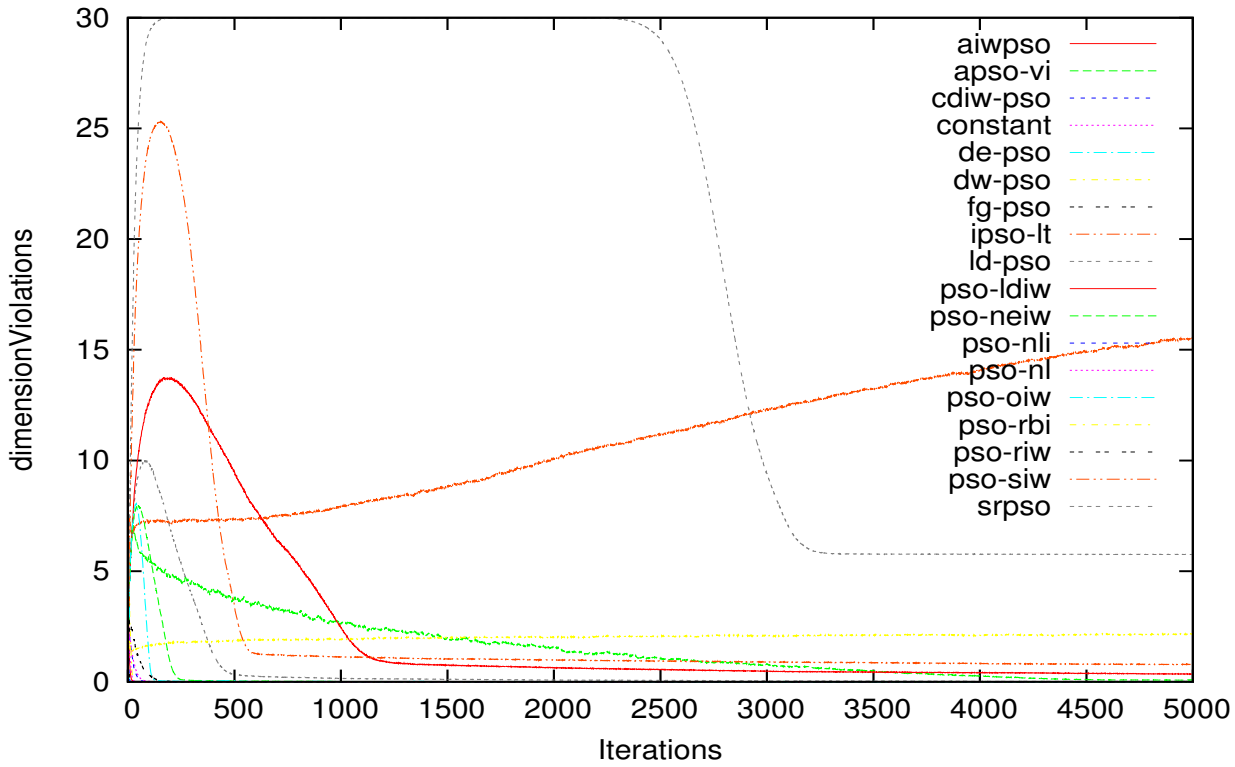




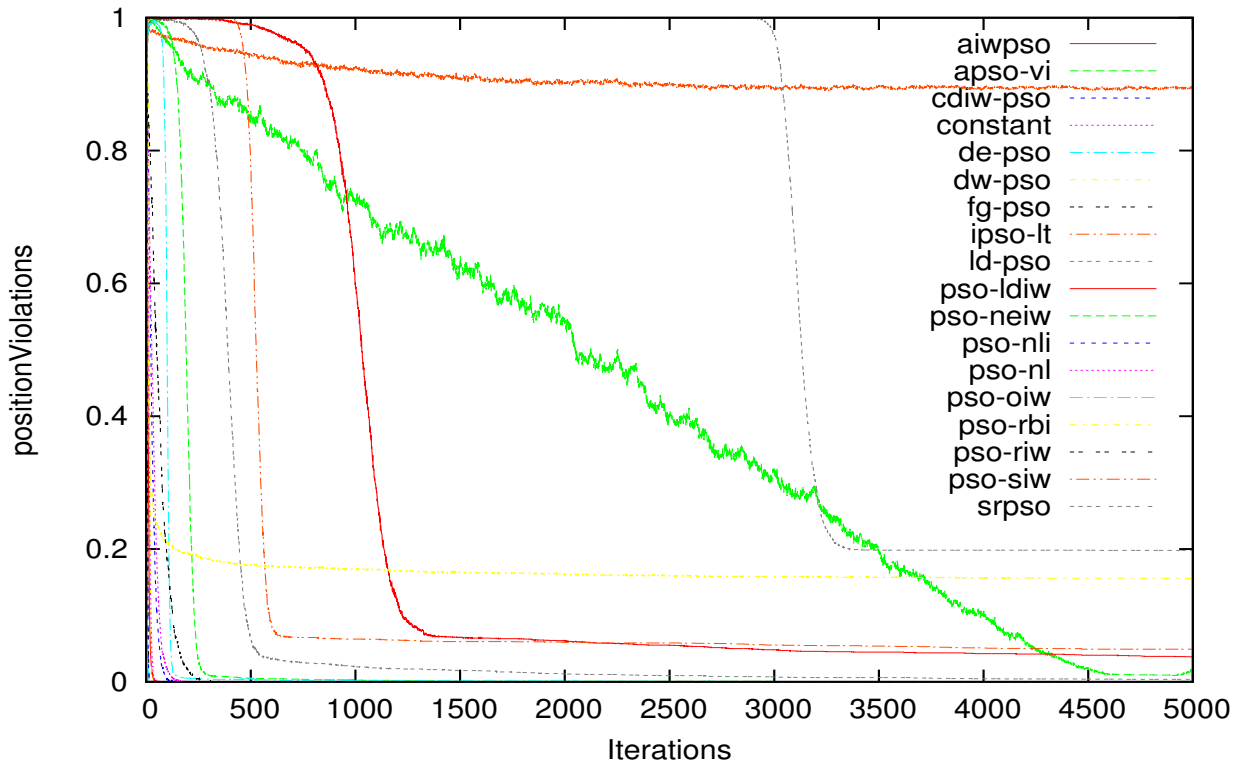
Average swarm diversity



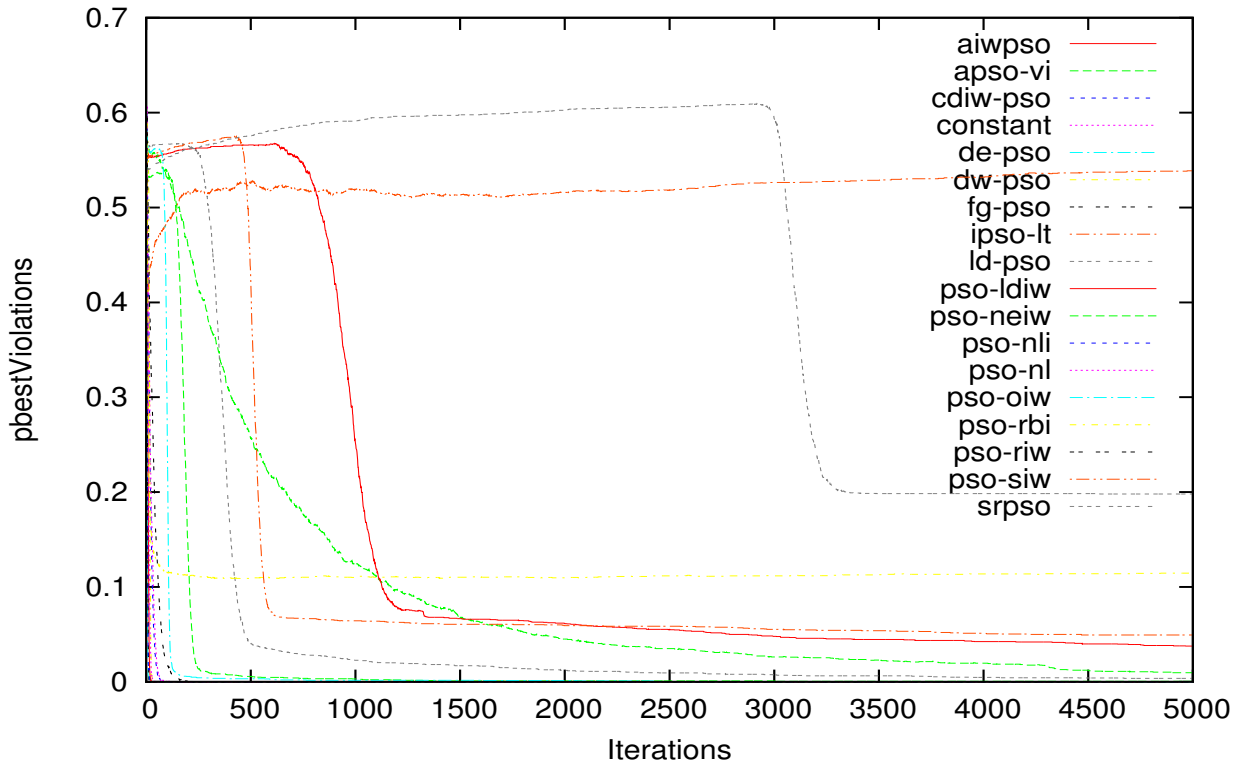
Average boundary violations per dimension



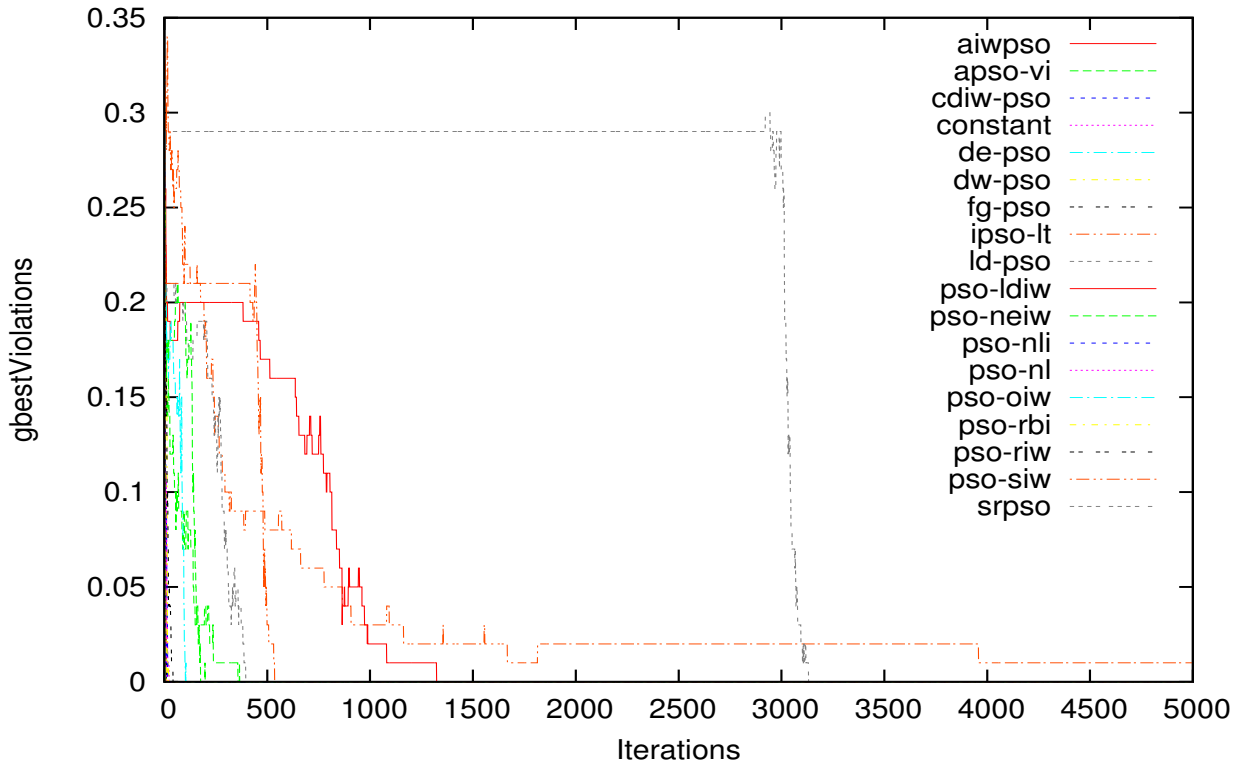
Average percentage particle position boundary violations



Average percentage personal best position boundary violations



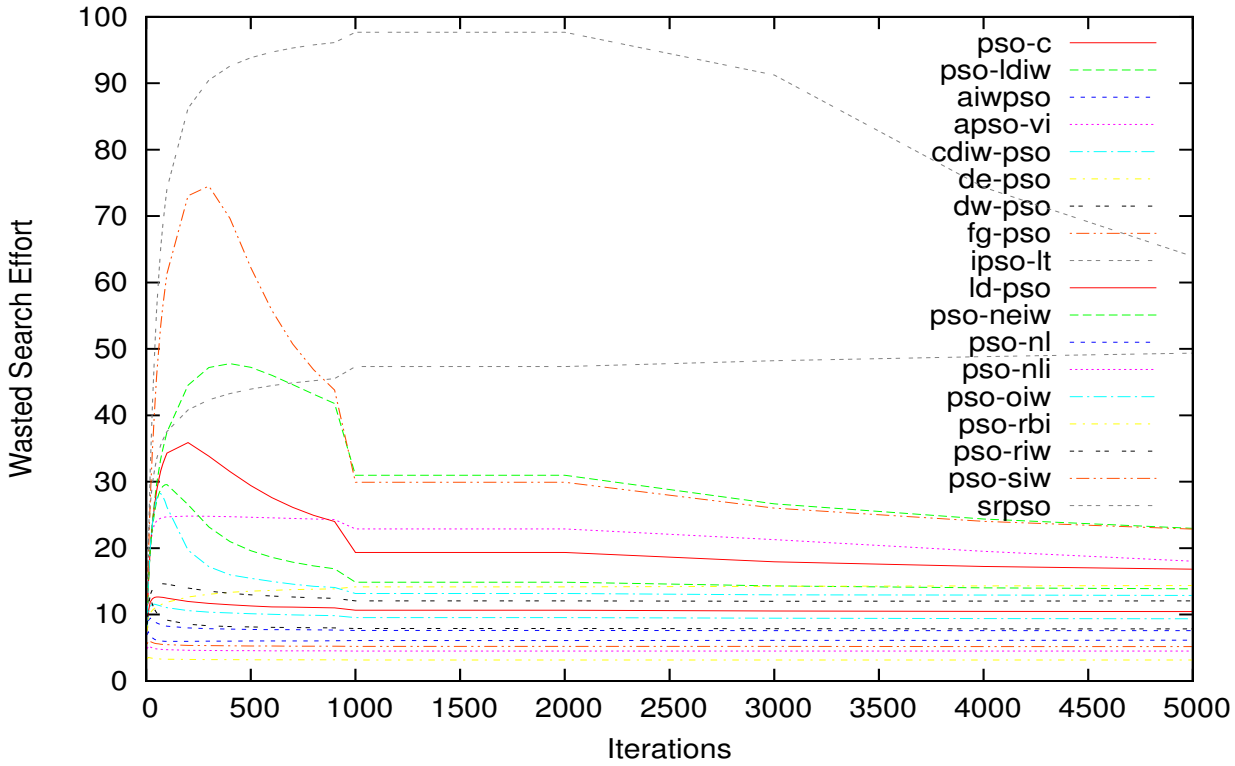
Average global best position boundary violations



Control Parameters

Self-Adaptive Particle Swarm Optimization: Ackley (cont)

Wasted search effort over 60 functions, in dimensions 30, 40, 50, 60, 80, 90, and 100



Uses the specially-formulated function to study convergence behavior[11]:

$$F(\mathbf{x}) \sim U(0, 2000)$$

such that

$$F(\mathbf{x}_1) = F(\mathbf{x}_2) \text{ if } \mathbf{x}_1 = \mathbf{x}_2$$

- the fitness value of each position in the search space is randomly sampled within the range $[0, 2000]$
- complete stagnation is highly unlikely
- provides a good benchmark function for studying convergence behavior

[11] CW Cleghorn, AP Engelbrecht, *Particle Swarm Variants: Standardized Convergence Analysis*, Swarm Intelligence, 9(2-3):177–203, 2015

Control Parameters

Self-Adaptive Particle Swarm Optimization: Analysis (cont)

Performance measures:

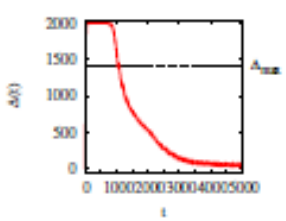
- Average particle movement, Δ:
  - quantifies average particle step size
  - if value does not decrease, particles do not converge
- Percentage particles with convergent control parameters, *CP*:
  - measures algorithm's ability to generate convergent parameters
- Average parameter movement, Δ_p:
  - average step size in parameter space
  - quantifies stability of the control parameter values
- Percentage particles that violates boundaries, *IP*:
  - proportion of particles that violates boundary constraints in at least one dimension
  - quantification of wasted search effort

After 5000 interations:

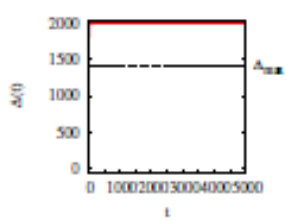
Algorithm	Δ	CP	Δ_p	IP
PSO	415.125	100%	0.0	70.7%
PSO-TVIW	56.489	100%	1.00e-4	9.6%
PSO-AIWF	2000.000	0%	0.0	96.7%
DAPSO	2000.000	0%	NaN	96.9%
IPSO-LT	2000.000	0%	0.0	96.7%
SAPSO-LFZ	2000.000	47.2%	0.0	53.5%
SAPSO-DWCY	1324.322	100%	0.0	96.2%
PSO-RBI	2000.000	76.7%	6.01e-2	41.5%
IPSO-CLL	2000.000	100%	0.0	100%
AIWPSO	45.521	100%	0.0	3.3%
APSO-VI	55.940	100%	0.0	6.1%
SRPSO	2000.000	96.7%	0.0	3.3%
PSO-SAIC	2000.000	0%	NaN	96.7%
PSO-RAC	165.544	100%	1.60e+0	44.2%
PSO-TVAC	32.354	100%	5.74e-4	6.5%
PSO-ICSA	2000.000	0%	4.00e-4	96.7%
APSO-ZZLC	1318.307	100%	4.51e-5	96.1%
UAPSO-A	124.467	70%	8.47e-1	38.1%
GPSO	2000.000	16.7%	8.35e-2	96.7%

Control Parameters

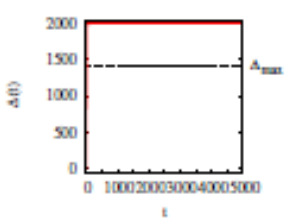
Self-Adaptive Particle Swarm Optimization: Average Particle Movement



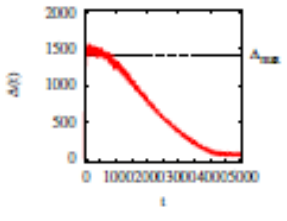
(a) PSO-TVIW



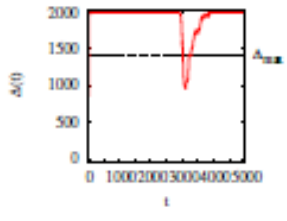
(b) PSO-AIWF



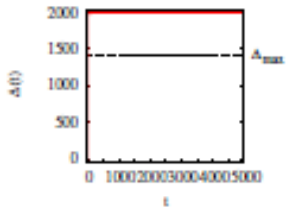
(c) DAPSO



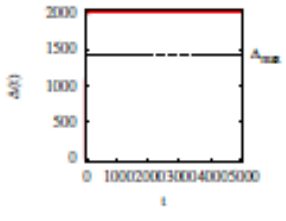
(j) APSO-VI



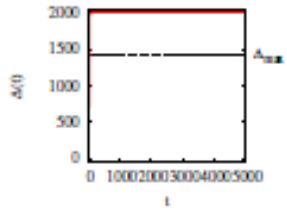
(k) SRPSO



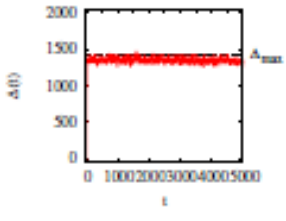
(l) PSO-SAIC



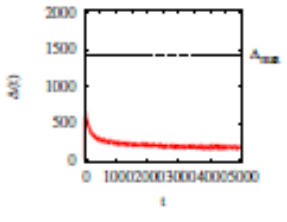
(d) IPSO-LT



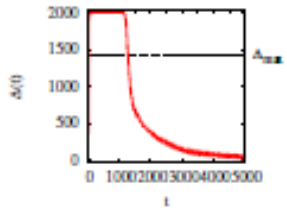
(e) SAPSO-LFZ



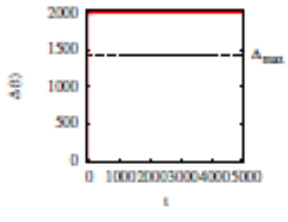
(f) SAPSO-DWCY



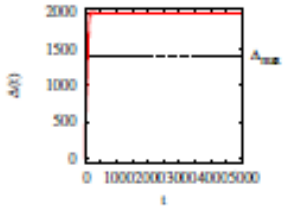
(m) PSO-RAC



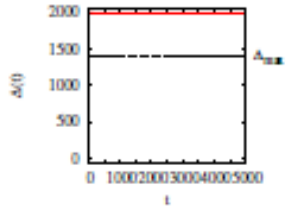
(n) PSO-TVAC



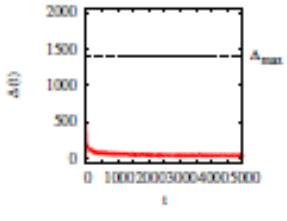
(o) PSO-ICSA



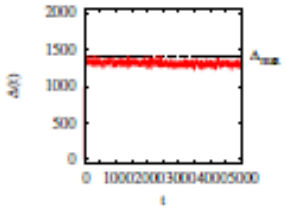
(g) PSO-RBI



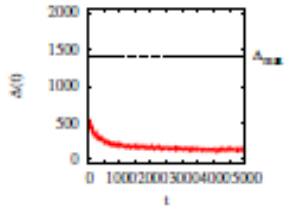
(h) IPSO-CLL



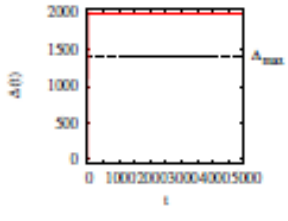
(i) AIWPSO



(p) APSO-ZZLC



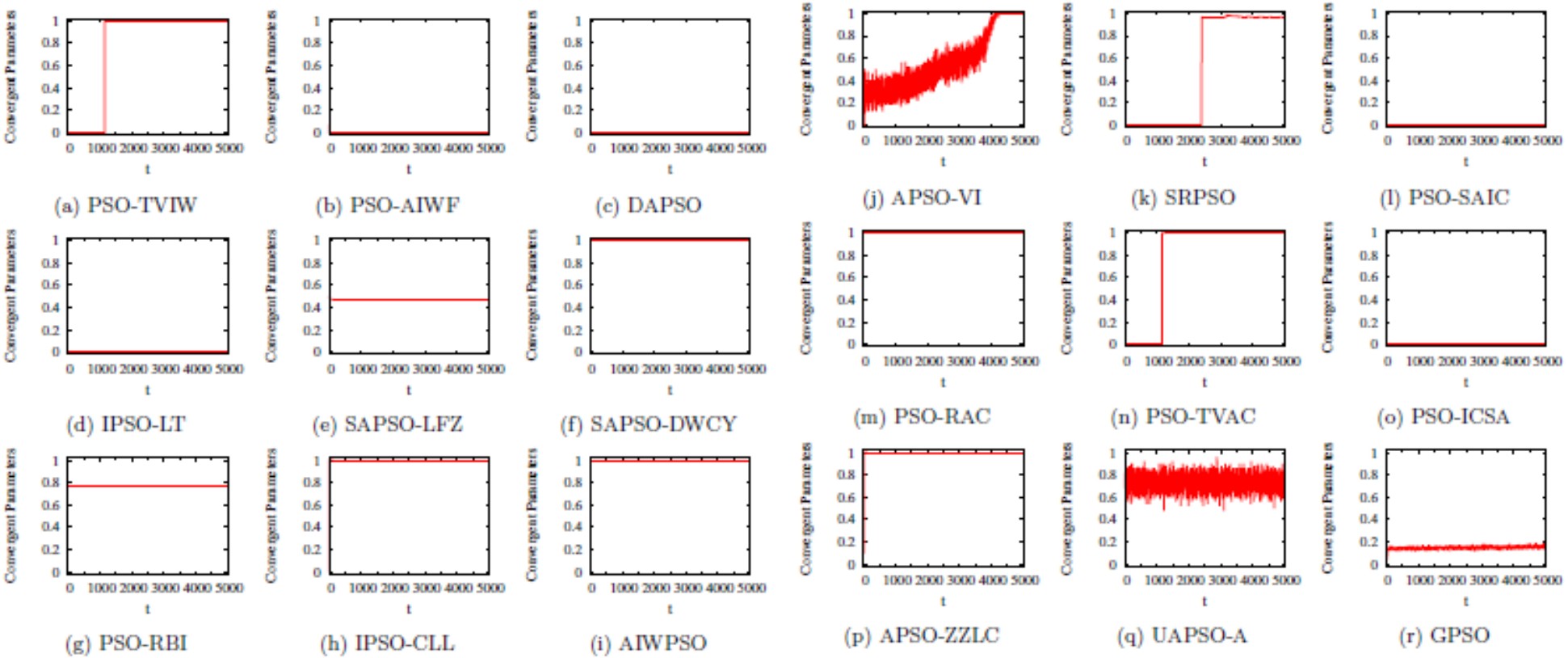
(q) UAPSO-A



(r) GPSO

Control Parameters

Self-Adaptive Particle Swarm Optimization:%Convergent Parameters



(a) PSO-TVIW (b) PSO-AIWF (c) DAPSO

(j) APSO-VI (k) SRPSO (l) PSO-SAIC

(d) IPSO-LT (e) SAPSO-LFZ (f) SAPSO-DWCY

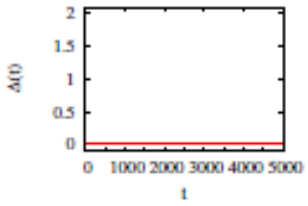
(m) PSO-RAC (n) PSO-TVAC (o) PSO-ICSA

(g) PSO-RBI (h) IPSO-CLL (i) AIWPSO

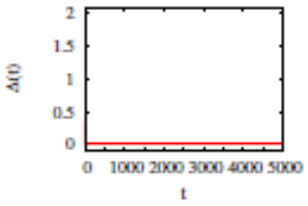
(p) APSO-ZZLC (q) UAPSO-A (r) GPSO

Control Parameters

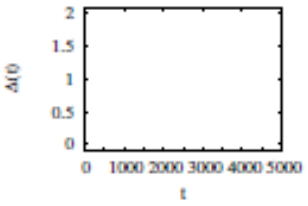
Self-Adaptive Particle Swarm Optimization: Parameter Movement



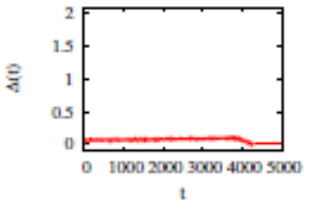
(a) PSO-TVIW



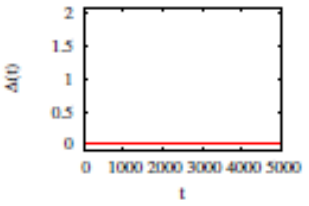
(b) PSO-AIWF



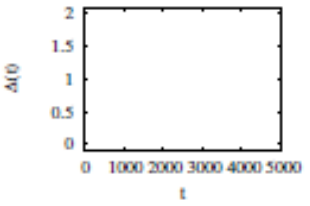
(c) DAPSO



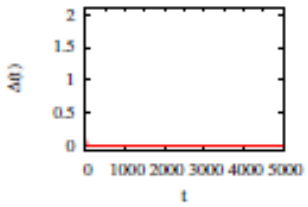
(j) APSO-VI



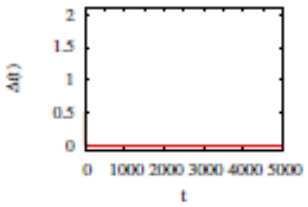
(k) SRPSO



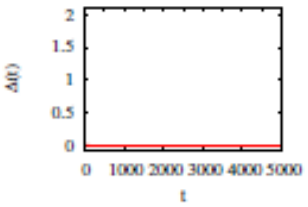
(l) PSO-SAIC



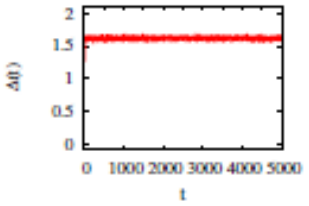
(d) IPSO-LT



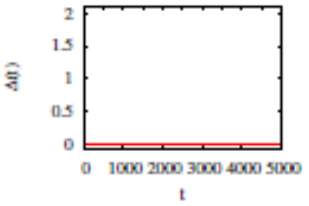
(e) SAPSO-LFZ



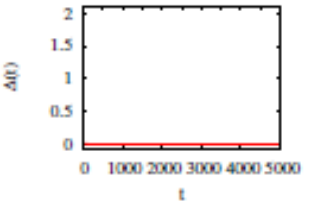
(f) SAPSO-DWCY



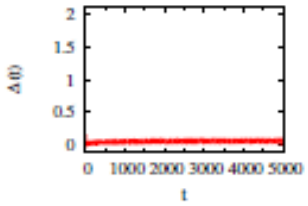
(m) PSO-RAC



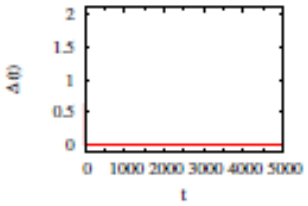
(n) PSO-TVAC



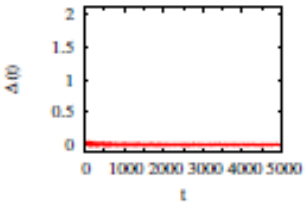
(o) PSO-ICSA



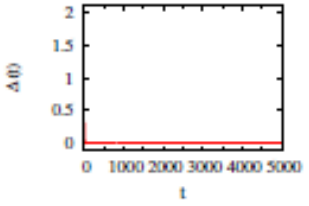
(g) PSO-RBI



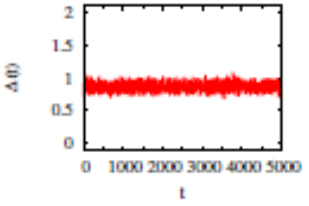
(h) IPSO-CLL



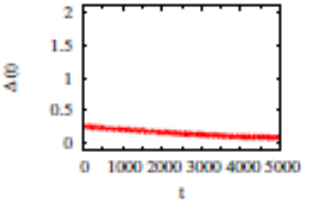
(i) AIWPSO



(p) APSO-ZZLC



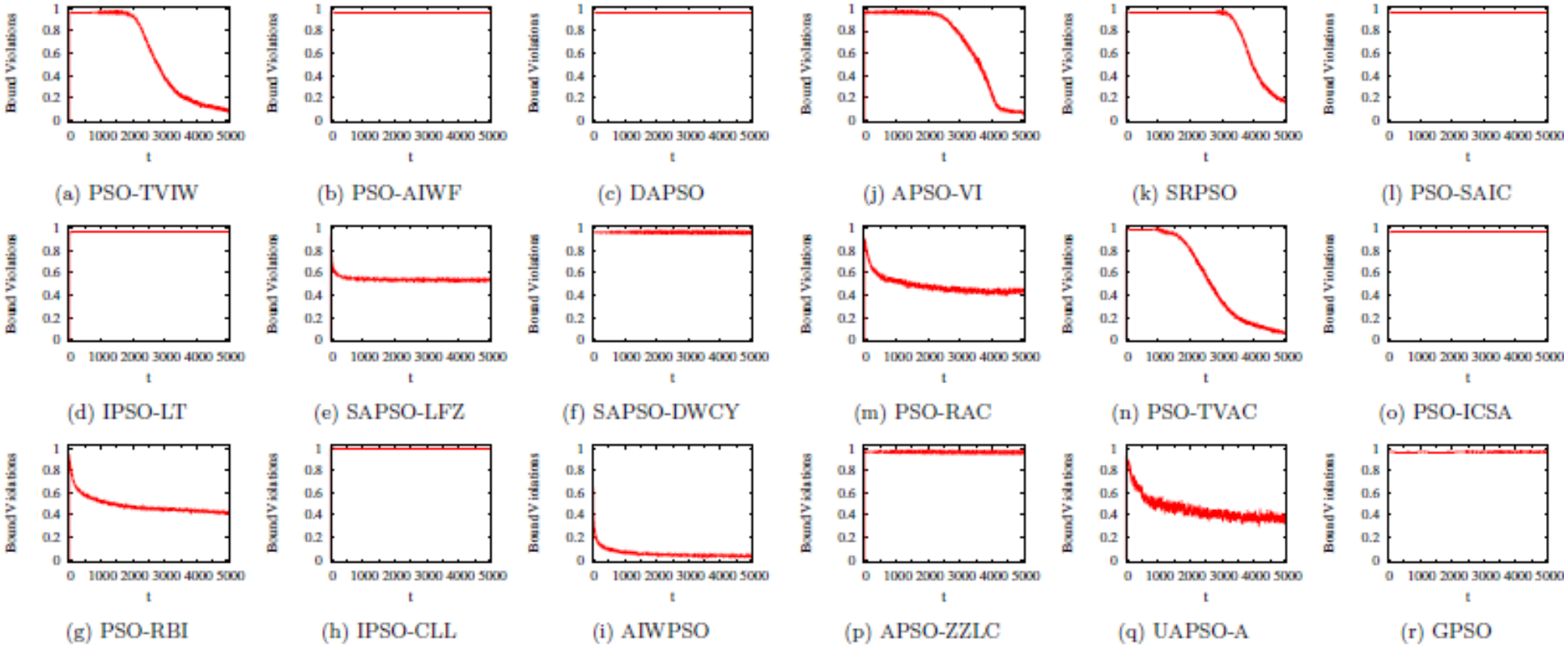
(q) UAPSO-A



(r) GPSO

Control Parameters

Self-Adaptive Particle Swarm Optimization: Boundary Violations



(a) PSO-TVIW (b) PSO-AIWF (c) DAPSO (j) APSO-VI (k) SRPSO (l) PSO-SAIC

(d) IPSO-LT (e) SAPSO-LFZ (f) SAPSO-DWCY (m) PSO-RAC (n) PSO-TVAC (o) PSO-ICSA

(g) PSO-RBI (h) IPSO-CLL (i) AIWPSO (p) APSO-ZZLC (q) UAPSO-A (r) GPSO

Control Parameters

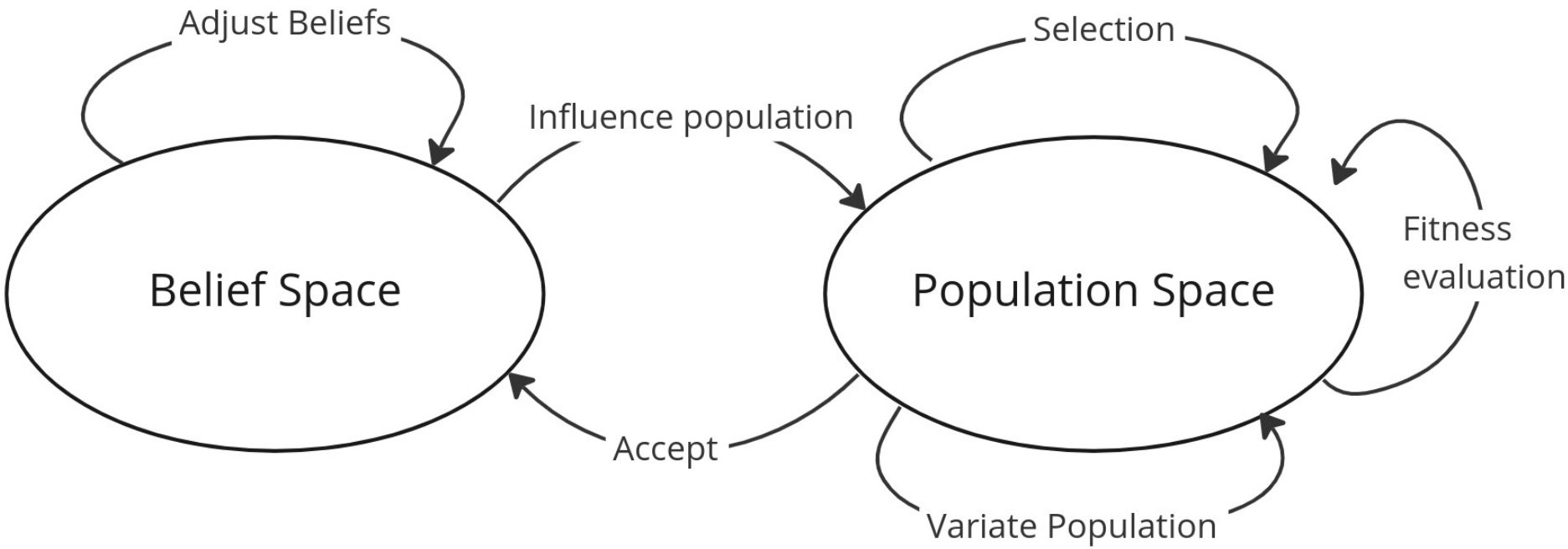
Stability-Guided Particle Swarm Optimization

sc1: Satisfies Van den Bergh & Engelbrecht stability condition
sc2: Satisfies Poli & Broomhead stability condition
s: $w = 0.7, c_1 = c_2 = 1.4$; t: tuned control parameter values

Function	PSO_s	PSO_t	PSO_{sc1}	PSO_{sc2}
Ackley 1	3	1	2	2
Alpine 1	4	1	2	3
Bohachevsky 1	3	1	2	2
BonyadiMichalewicz	2	1	1	1
Brown	4	3	2	1
CosineMixture	3	2	1	1
CrossInTray	4	1	3	2
Discus	4	1	3	2
DropWave	4	1	3	2
Easom	3	1	3	2
Elliptic	4	1	3	2
EggCrate	4	1	3	2
EggHolder	3	1	2	2
Exponential	2	1	1	1
Giunta	3	1	2	2
Levy 3	3	1	2	2
LevyMontalvo	3	1	2	2
Mishra 1	4	3	1	2
Mishra 4	3	1	2	2
Mishra 7	3	1	1	2
NeedleEye	1	1	1	1
Norwegian	2	3	1	4
Paviani	4	3	1	2
Penalty 1	4	1	3	2
Penalty 2	3	1	2	2
Pinter 2	3	1	2	2
Price 2	1	1	1	1
Qings	2	1	1	1
Quadric	2	1	3	4
Rana	4	1	3	2
Rastrigin	4	1	3	2
Ripple 25	4	2	3	1
Rosenbrock	2	1	1	1
Salomon	2	1	1	1
Schwefel 1	4	1	3	2
Schwefel 2.26	3	1	2	1
Shubert 4	4	2	3	1
Step 3	2	1	1	1
Trigonometric	4	1	3	2
Vincent	2	1	1	1
Weierstrass	3	4	1	2
XinSheYang 1	3	1	2	2
XinSheYang 3	3	2	1	2
XinSheYang 4	3	2	1	1
Average	3.02	1.37	1.96	1.83
Deviation	0.91	0.74	0.84	0.77

Belief Space Self-Adaptive Particle Swarm Optimization[12]

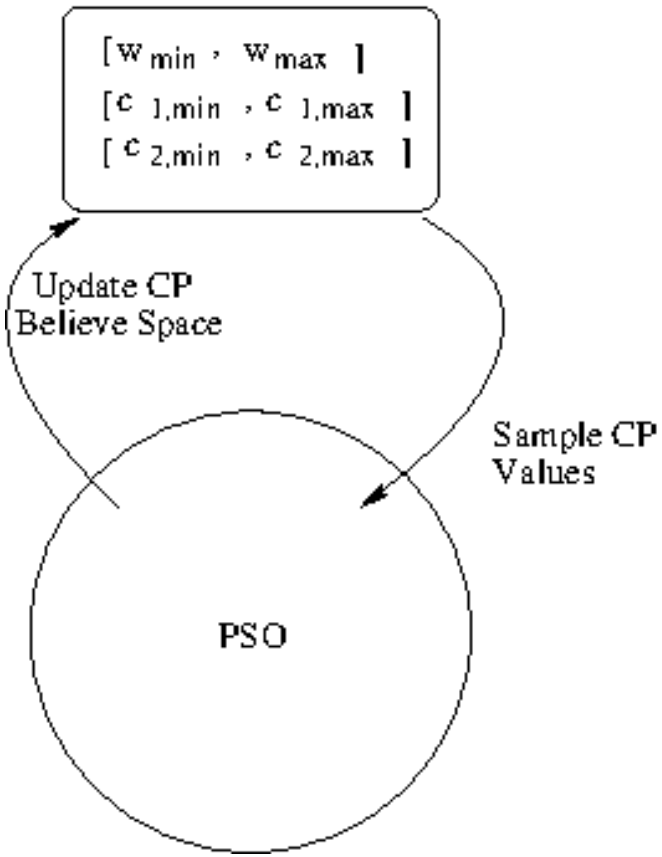
Cultural algorithms:



[12] DH von Eschwege, AP Engelbrecht, *Belief Space Guided Approach to Self-Adaptive Particle Swarm Optimization*, Swarm Intelligence, 2024

Belief Space Self-Adaptive Particle Swarm Optimization

The Approach



Algorithm aspects:

- How to select particles to update the control parameter belief space?
- How to update the control parameter belief space?
- When to update the control parameter belief space?
- How do particles sample the control parameter belief space?
- When do particles sample the control parameter belief space?

Belief Space Self-Adaptive Particle Swarm Optimization

The Approach (cont)

How to select particles to update the control parameter belief space?

- Random selection
- Elitist selection
- Roulette wheel selection
- Rank-based selection
- Improvement-based selection
- Tournament selection

Belief Space Self-Adaptive Particle Swarm Optimization

The Approach (cont)

When to update the control parameter belief space?

- Every iteration
- Every n_t iterations
- When global best position stagnates for too long

When to sample new control parameter values?

- As above

How to update the control parameter belief space?

- Minimum and maximum values of parameter ranges set to that of the selected particles

Belief Space Self-Adaptive Particle Swarm Optimization

The Approach (cont)

How do particles sample the control parameter belief space?

- Uniform randomly from the control parameter ranges
- Randomly to satisfy the stability conditions

Belief Space Self-Adaptive Particle Swarm Optimization

Empirical Process

The initial belief space:

- $w \in [0, 1]$
- $c_1, c_2 \in [0, 4]$

Other experimental parameters:

- Number of independent runs: 30
- Number of minimization problems: 31
- Problem dimensionality: 30
- Swarm size: 30 particles
- Number of iterations: 5000

Belief Space Self-Adaptive Particle Swarm Optimization

Empirical Process (cont)

Performance measures:

- Normalized objective function values of global best positions
- Convergence in belief space, i.e. control parameter space

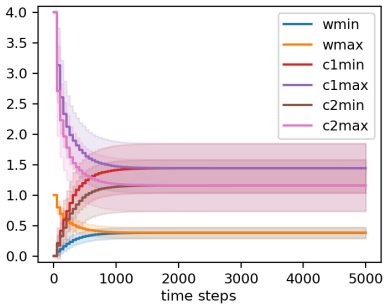
Performance baselines:

- Tuned global best inertia PSO (PSO-IW)
- With dynamic changing inertia weight (PSO-TVIW)
- With dynamic changing acceleration coefficients (PSO-TVAC)
- Sample random values from stability region (PSO-RAC)

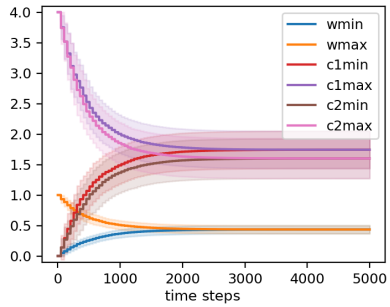
Belief Space Self-Adaptive Particle Swarm Optimization

Results

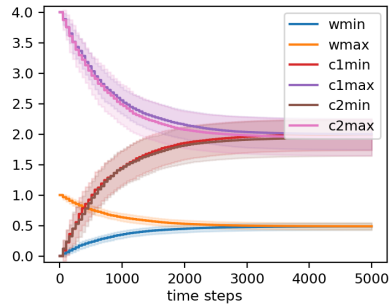
Belief space boundaries for improvement-based selection
After every 50 iterations



(a) $n_e = 10$

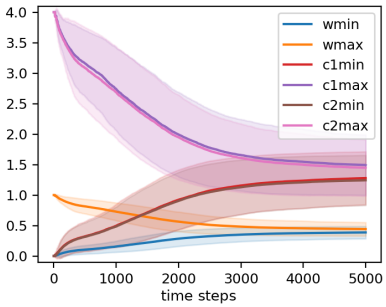


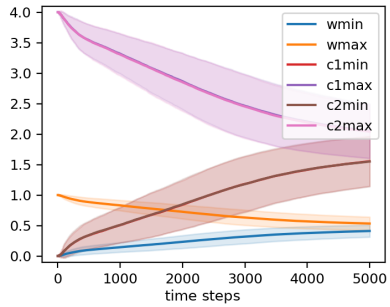
(b) $n_e = 20$

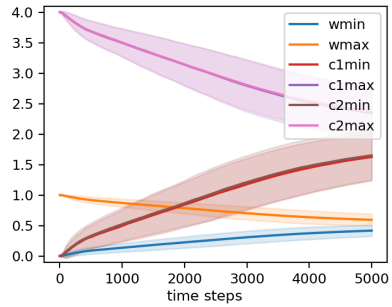


(c) $n_e = 30$

After stagnation for 20 iterations



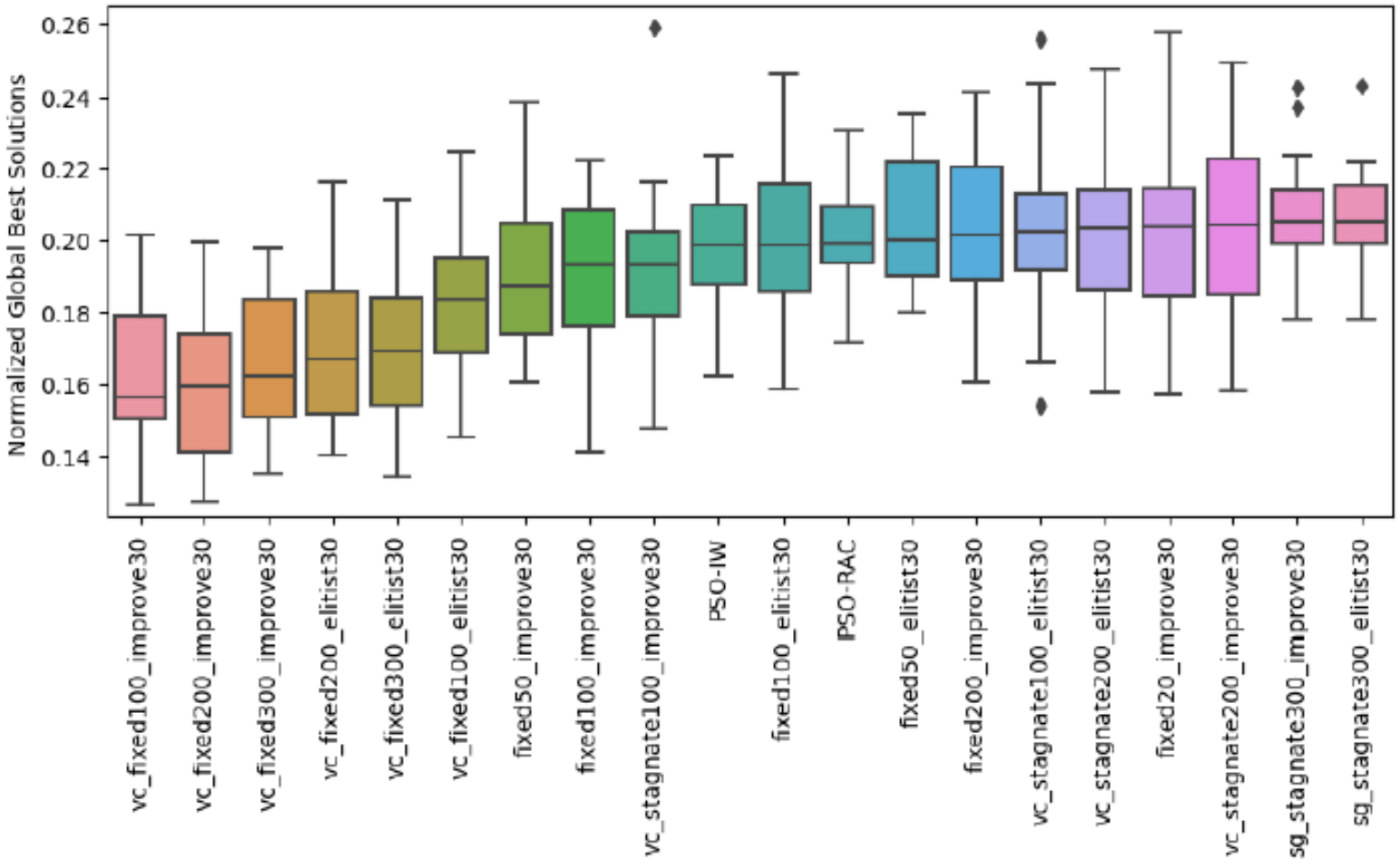




Belief Space Self-Adaptive Particle Swarm Optimization

Results (cont)

Normalized global best solution quality



Belief Space Self-Adaptive Particle Swarm Optimization

Results (cont)

The main findings:

- Selection methods with higher selective presure such as elitist and improvement-based selection perform better
- Larger values for the number of particles allowed to update the belief space, and waiting longer before updating control parameter values result in slower convergence in the belief space, with generally better performance
- Best performance is when belief space is updated at fixed intervals for improvement-based selection
- Velocity clamping further improved performance
- Best performing BS-SAPSO showed improvement of 20% in solution quality compared to the best baseline, i.e. PSO-IW

Heterogeneous Particle Swarm Optimization[13]

Introduction

Particles in standard particle swarm optimization (PSO), and most of its modifications, follow the same behavior:

- particles implement the same velocity and position update rules
- particles therefore exhibit the same search behaviours
- the same exploration and explotation abilities are achieved

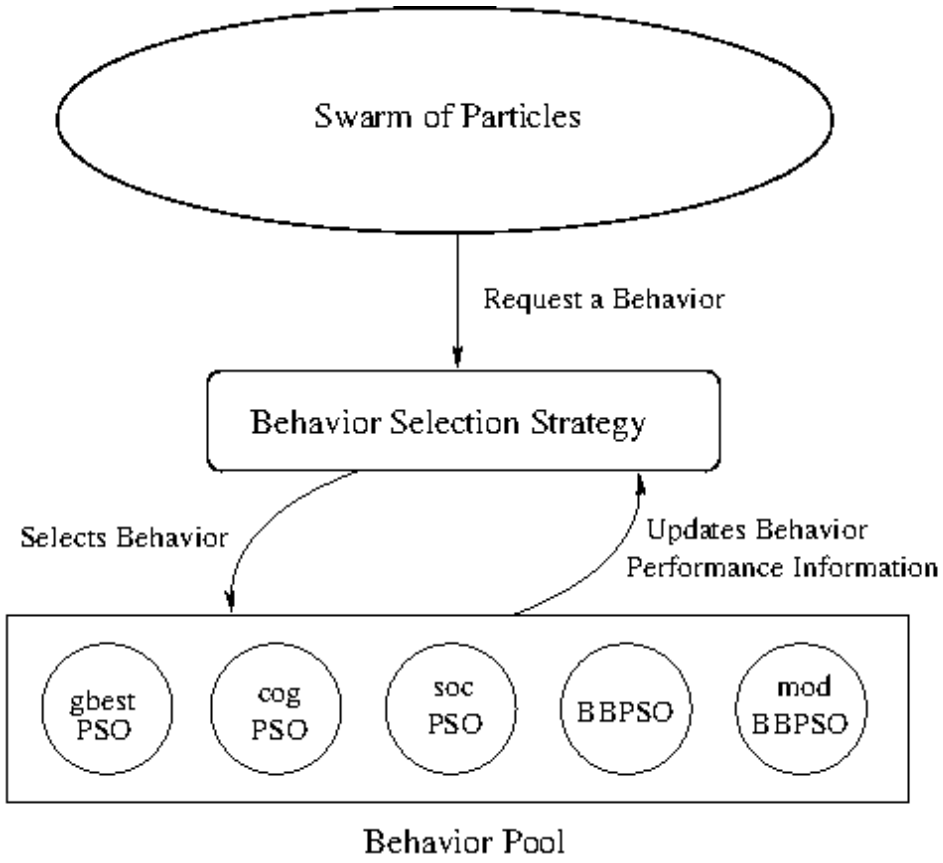
Heterogeneous swarms contain particles that follow different behaviors:

- particles follow different velocity and position update rules
- some particles may explore longer than others, while some may exploit earlier than others
- a better balance of exploration and exploitation can be achieved provided a pool of different behaviors is used

[13] AP Engelbrecht, *Heterogeneous Particle Swarm Optimization*, Seventh International Conference on Swarm Intelligence, 2010

Heterogeneous Particle Swarm Optimization

The Model



A behavior is defined as both

- the velocity update, and
- the position update

of a particle

Requirements for behaviors in the behavior pool:

- must exhibit different search behaviors
- different exploration-exploitation phases
- that is, different exploration-exploitation finger prints

Exploration-exploitation finger print determined through diversity profile

Diversity calculated as

$$\mathcal{D} = \frac{1}{n_s} \sum_{i=1}^{n_s} \sqrt{\sum_{j=1}^{n_x} (x_{ij} - \overline{x}_j)^2}$$

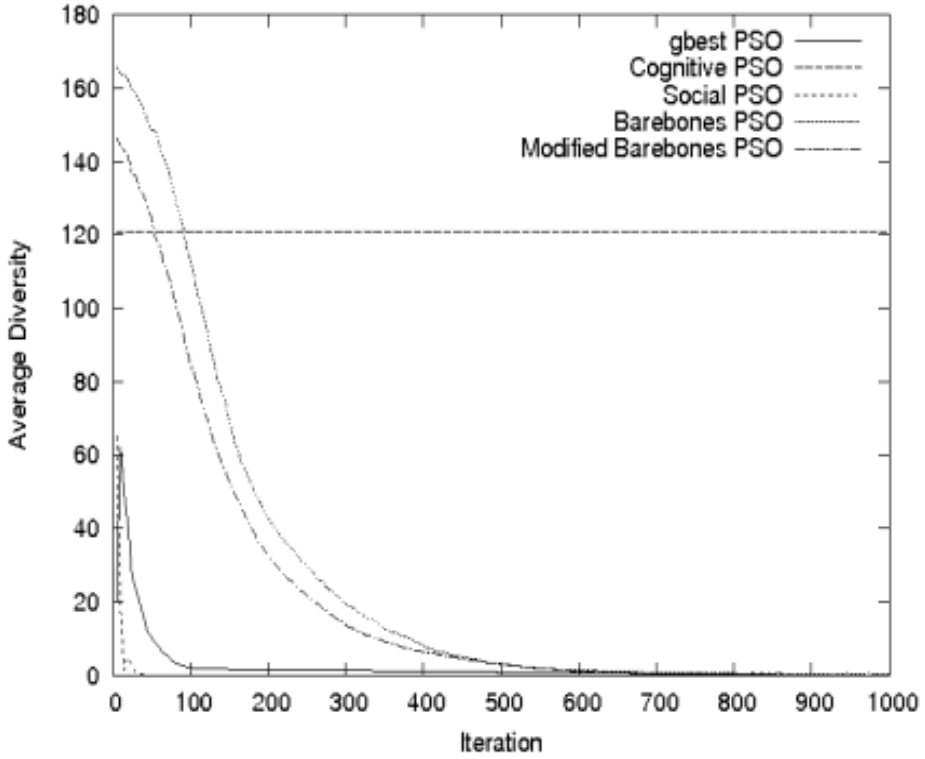
where the swarm center is

$$x_j = \frac{\sum_{i=1}^{n_s} x_{ij}}{n_s}$$

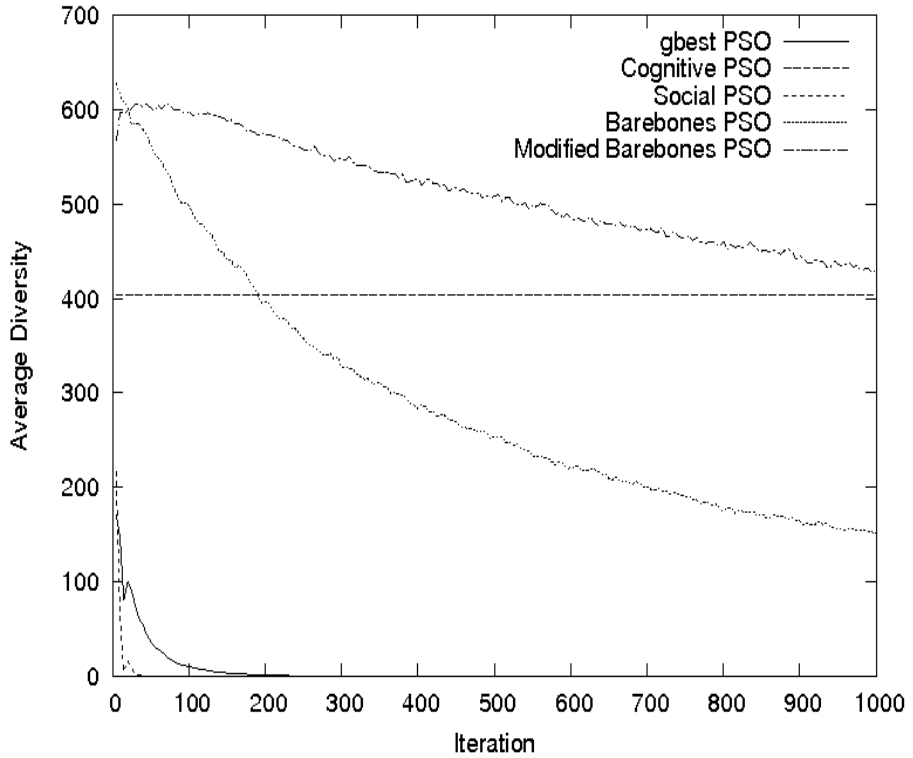
and n_s is the number of particles

Heterogeneous Particle Swarm Optimization

Exploration-Exploitation Finger Prints (cont)



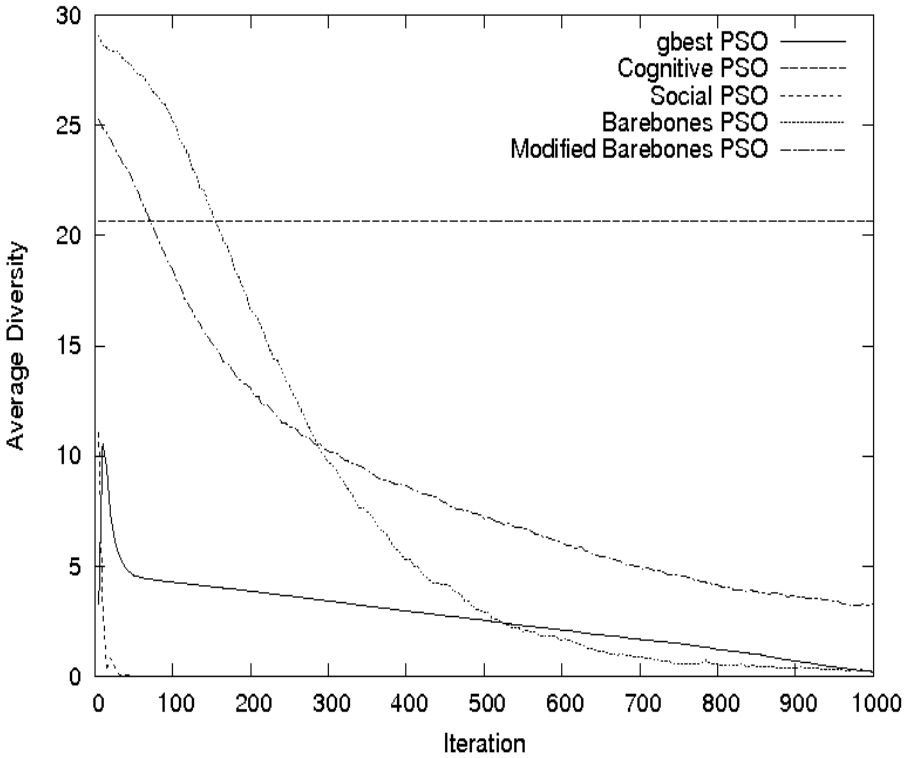
(a) Ackley



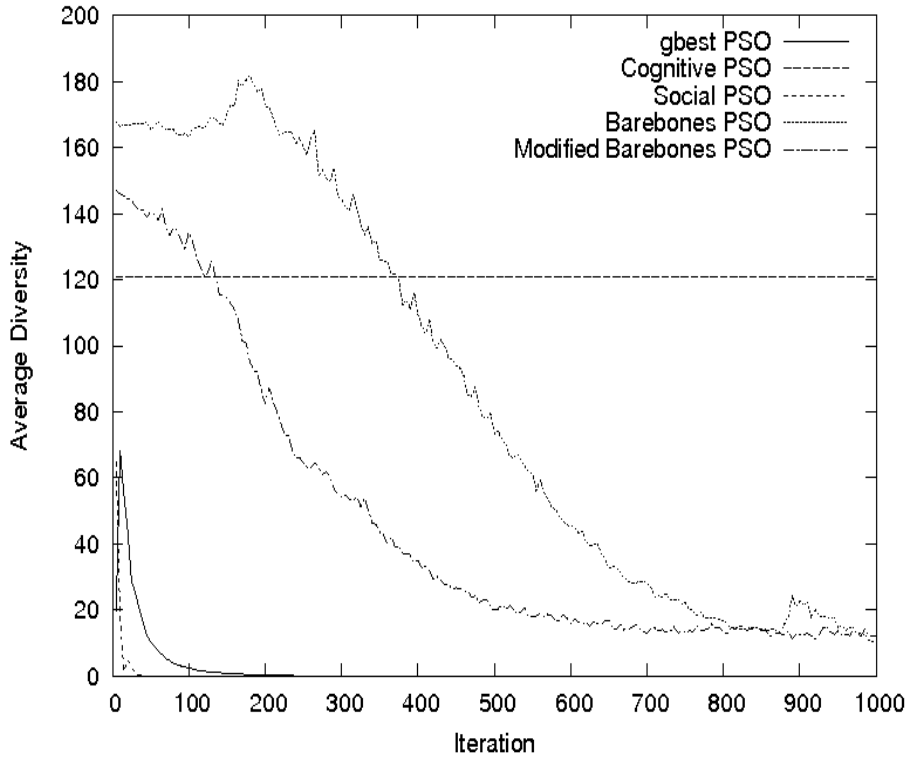
(b) Quadric

Heterogeneous Particle Swarm Optimization

Exploration-Exploitation Finger Prints (cont)



(c) Rastrigin



(d) Rosenbrock

Heterogeneous Particle Swarm Optimization

Review

A number of PSO algorithms do already exist which allow particles to follow different search behaviors:

- Division of labor PSO: Particles are allowed to switch to local search near the end of the search process
- Life-cycle PSO: Particles follow a life-cycle, changing from a PSO particle, to GA individual, to a hill-climber
- Predator-prey PSO: Predator particles are attracted to only the gbest position, prey particles follow the standard velocity update rule
- Guaranteed convergence PSO (GCPSO): Global best particle follows a different, exploitaitve search around best position, while other particles follow normal velocity updates

Heterogeneous Particle Swarm Optimization

Review (cont)

- NichePSO: Main swarm of particles folllow cognitive PSO, while sub-swarms follow GCPSO behavior
- Charged PSO: Charged particles adds a repelling force to the velocity update
- Heterogeneous cooperative PSO: Sub-swarms use different meta-heuristics

Two different initial HPSO models:

- **static** HPSO (SHPSO)
  - behaviors randomly assigned from behavior pool during initialization
  - behaviors do not change
- **dynamic** HPSO (DHPSO)
  - behaviors are randomly assigned
  - may change during search
  - when particle stagnates, it randomly selects a new behavior
  - a particle stagnates if its personal best position does not change over a number of iterations

Heterogeneous Particle Swarm Optimization

Two HPSO Models: Results

Algorithm	Ack	Quad	Ras	Ros	Sal	Grie	Average
	10 Dimensions						
Standard PSO	5	5	5	5	5	5	5
Social PSO	6	6	6	6	6	6	6
Cognitive PSO	7	7	7	7	7	7	7
Barebones PSO	4	1	4	4	3	3	3.17
Modified Barebones	2	4	1	2	2	1	2
Static HPSO	3	2	2	1	4	4	2.67
Dynamic HPSO	1	3	3	3	1	2	2.17
	30 Dimensions						
Standard PSO	5	4	4	5	5	5	4.67
Social PSO	6	5	3	6	6	6	5.33
Cognitive PSO	7	7	6	7	7	7	6.83
Barebones PSO	2	3	7	3	3	3	3.5
Modified Barebones	3	6	1	4	2	1	2.83
Static HPSO	4	2	5	1	4	4	3.33
Dynamic HPSO	1	1	2	2	1	2	1.5
	50 Dimensions						
Standard PSO	5	3	3	4	5	5	4.17
Social PSO	6	4	4	5	6	6	5.17
Cognitive PSO	7	7	6	7	7	7	6.83
Barebones PSO	3	5	2	6	3	3	3.67
Modified Barebones	2	6	7	3	4	2	4
Static HPSO	4	2	5	2	2	4	3.17
Dynamic HPSO	1	1	1	1	1	1	1
	100 Dimensions						
Standard PSO	5	3	3	3	3	4	3.5
Social PSO	6	4	4	4	4	5	4.5
Cognitive PSO	7	7	2	5	6	7	5.67
Barebones PSO	4	5	6	6	5	6	5.33
Modified Barebones	3	6	7	7	7	3	5.5
Static HPSO	2	2	5	2	2	2	2.5
Dynamic HPSO	1	1	1	1	1	1	1

	Average over all Dimensions						
Standard PSO	5	3.75	3.75	4.25	4.5	4.75	4.33
Social PSO	6	4.75	4.25	5.25	5.5	5.75	5.25
Cognitive PSO	7	7	5.25	6.5	6.75	7	6.58
Barebones PSO	3.25	3.5	4.75	4.75	3.5	3.75	3.92
Modified Barebones	2.5	5.5	4	4	3.75	1.75	3.58
Static HPSO	3.25	2	4.25	1.5	3	3.5	2.92
Dynamic HPSO	1	1.5	1.75	1.75	1	1.5	1.42

What is the problem?

- Current HPSO models do not make use of any information about the search process to guide selection towards the most promising behaviors

What is the solution?

- An approach to self-adapt behaviors, i.e. to select the best behaviors probabilistically based on information about the search process

Heterogeneous Particle Swarm Optimization

Self-Adaptive HPSO: Review[14]

Related self-adaptive approaches to HPSO algorithms:

- Difference proportional probability PSO (DPP-PSO)
  - particle behaviors change to the nbest particle behavior
  - based on probability proportional to how much better the nbest particle is
  - includes static particles for each behavior, i.e. behaviors do not change
  - only two behaviors, i.e. FIPS and original PSO
- Adaptive learning PSO-II (ALPSO-II)
  - Behaviors are selected probabilistically based on improvements that they affect to the quality of the corresponding particles
  - Overly complex and computationally expensive

[14] F. Nepomuceno and A.P. Engelbrecht, *A Self-Adaptive Heterogeneous PSO Inspired by Ants*, International Swarm Intelligence Conference, 2012; FV Nepomuceno, AP Engelbrecht, *A self-adaptive heterogeneous PSO for real parameter optimization*, IEEE Congress on Evolutionary Computation, 2013

Heterogeneous Particle Swarm Optimization

Self-Adaptive HPSO: General Structure

```
initialize swarm
while stopping conditions not met do
    for each particle do
        if change schedule is triggered then
            select new behavior for the particle
        end if
    end for
    for each particle do
        update particle's velocity and position based on selected behavior
    end for
    for each particle do
        update pbest and gbest
    end for
    for each behavior do
        update behavior score/desirability
    end for
end while
```

Expanded behavior pool: That of dHPSO, plus

- Quantum PSO (QPSO)
- Time-varying inertia weight PSO (TVIW-PSO)
- Time-varying acceleration coefficients (TVAC-PSO)
- Fully informed particle swarm (FIPS)

Two self-adaptive strategies inspired by foraging behavior of ants

- Ants are able to find the shortest path between their nest and a food source
- Paths are followed probabilistically based on pheromone concentrations on the paths

Definitions:

- B is the total number of behaviors
- b is a behavior index
- p_b is the pheronome concentration for behavior b
- $prob_b(t)$ is the probability of selecting behavior b at time step t

$$prob_b(t) = \frac{p_b(t)}{\sum_{i=1}^{B} p_i(t)}$$

Each particle selects a new behavior, using Roulette wheel selection, when a behavior change is triggered

- Constant strategy (pHPSO-const):
  - Rewards behaviors if they improve or maintain a particle's fitness regardless the magnitude of improvement

$$p_b(t) = p_b(t) + \sum_{i=1}^{S_b} \begin{cases} 1.0 & \text{if } f(x_i(t)) < f(x_i(t-1)) \\ 0.5 & \text{if } f(x_i(t)) = f(x_i(t-1)) \\ 0.0 & \text{if } f(x_i(t)) > f(x_i(t-1)) \end{cases}$$

  - S_b is the number of particles using behavior b

- Linear strategy (pHPSO-lin):
  - Behaviors are rewarded proportional to the improvement in particle fitness

$$p_b(t) = p_b(t) + \sum_{i=1}^{s_b} \left(f\left(x_i(t-1)\right) - f\left(x_i(t)\right) \right)$$

  - A lower bound of 0.01 is set for each p_b to prevent zero or negative pheromone concentrations
  - Each behavior therefore always has a non-zero probability of being selected

To maintain diversity in the behavior pool, pheromone evaporates:

$$p_b(t+1) = \frac{\left(\sum_{i=1, i\neq b}^{B} p_i\right)}{\sum_{i=1}^{B} p_i} \times p_b$$

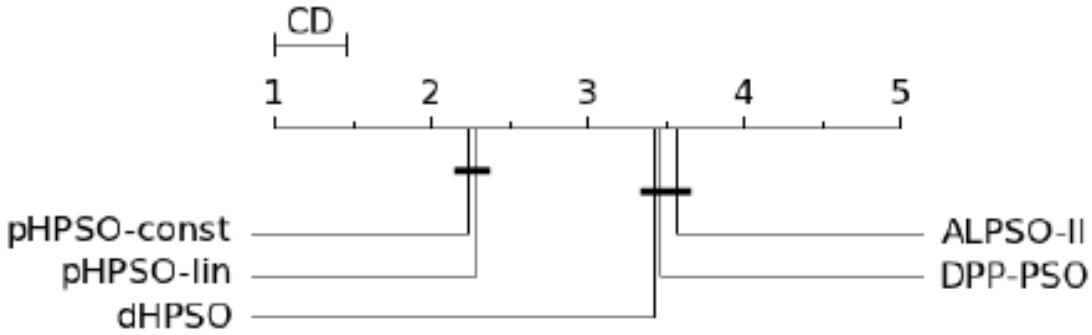
Amount of evaporation is proportional to the behavior's pheromone concentration as a ratio to the total pheromone concentration

A more desirable behavior has stronger evaporation to prevent domination

The pHPSO strategies are

- computationally less expensive than others,
- behaviors are self-adapted based on success of the corresponding behaviors, and
- better exploration of behavior space is achieved through pheromone evaporation
- introduces no new control parameters

Performance over 15 benchmark functions



Dimensions	ALPSO-II	DPP-PSO	dHPSO	pHPSO-const	pHPSO-lin
10	4.15±1.64	3.07±0.88	3.14±1.55	2.43±0.72	**2.21±1.15**
20	3.86±1.68	3.36±1.11	3.07±1.33	2.75±1.24	**1.96±0.77**
30	3.57±1.68	3.5±1.05	3.43±1.45	2.36±0.97	**2.14±1.06**
40	3.57±1.68	3.64±0.97	3.29±1.39	**2.07±0.88**	2.43±1.24
50	3.43±1.63	3.43±0.98	3.64±1.44	2.36±1.29	**2.14±0.83**
60	3.43±1.64	3.71±0.88	3.43±1.45	**2.0±0.93**	2.43±1.18
70	3.36±1.67	3.29±1.03	3.64±1.34	**2.21±1.32**	2.5±1.05
80	3.5±1.68	3.29±0.88	3.71±1.39	**2.0±0.85**	2.5±1.3
90	3.36±1.67	3.5±0.91	3.79±1.42	**2.0±1.07**	2.35±0.89
100	3.5±1.68	3.43±0.82	3.57±1.45	2.29±1.28	**2.21±0.94**
Mean	3.57±0.23	3.42±0.18	3.47±0.23	**2.24±0.23**	2.3±0.17

Heterogeneous Particle Swarm Optimization

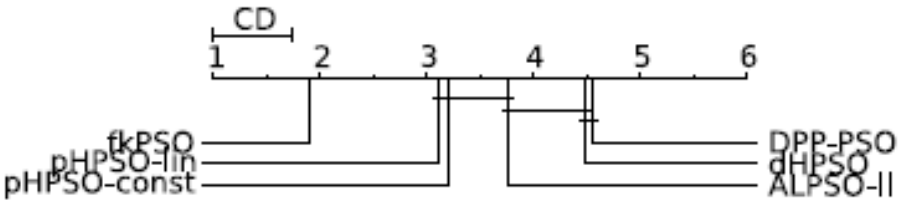
Self-Adaptive HPSO: Frequency-Based

Frequency-based HPSO is based on the premise that behaviors are more desirable if they frequently perform well:

- Each behavior has a success counter
- Success counter keeps track of the number of times that the behavior improved the fitness of a particle
- Only the successes of the previous k iterations are considered, so that behaviors that performed well initially, and bad later, do not continue to dominate in the selection process
- Behaviors change when a particle's pbest position stagnates
- Next behavior chosen using tournament selection
- Two new control parameters: k and tournament size

Heterogeneous Particle Swarm Optimization

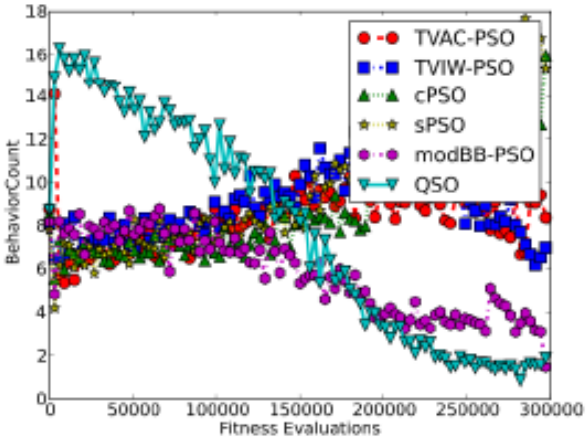
Self-Adaptive HPSO: Frequency-Based Results



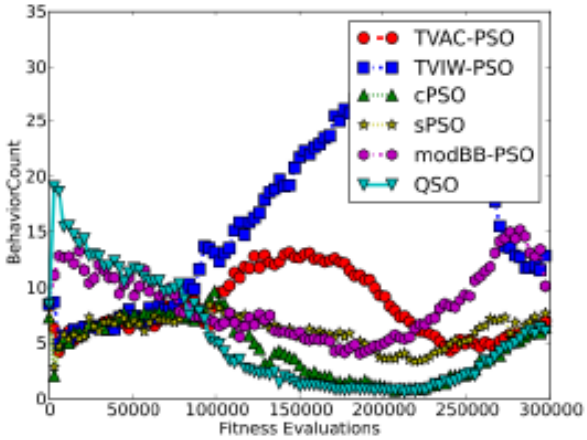
		ALPSO-II	dHPSO	DPP-PSO	f_k-PSO	pHPSO-const	pHPSO-lin
Unimodal	f_1	4.00	**3.00**	5.00	**3.00**	**3.00**	**3.00**
	f_2	3.00	5.67	**1.33**	3.00	3.33	4.67
	f_3	6.00	4.33	1.67	**1.33**	4.33	3.33
	f_4	6.00	5.00	4.00	**1.67**	2.33	2.00
	f_5	5.67	**2.83**	4.00	**2.83**	**2.83**	**2.83**
	Mean	4.93	4.17	3.2	**2.37**	3.16	3.17
	Std	1.22	1.11	1.44	0.72	0.67	0.87
Basic Multimodal	f_6	**2.67**	4.00	**2.67**	3.33	3.67	4.67
	f_7	4.33	2.00	5.67	**1.00**	3.67	4.33
	f_8	3.00	5.00	2.33	**1.33**	4.33	5.00
	f_9	5.00	6.00	4.00	**1.00**	2.33	2.67
	f_{10}	3.33	4.00	4.33	**3.00**	**3.00**	3.33
	f_{11}	**1.00**	3.33	6.00	4.00	2.33	4.33
	f_{12}	6.00	2.67	5.00	**1.00**	3.00	3.33
	f_{13}	5.00	2.67	6.00	**1.00**	3.33	3.00
	f_{14}	**1.00**	5.00	6.00	2.67	4.00	2.33
	f_{15}	4.67	6.00	4.00	**1.00**	3.33	2.00
	f_{16}	2.00	6.00	4.33	**1.00**	3.33	4.33
	f_{17}	**1.00**	4.67	6.00	2.67	4.33	2.33
	f_{18}	5.67	4.33	4.33	**1.00**	2.33	3.33
	f_{19}	**1.00**	4.67	6.00	3.67	3.33	2.33
	f_{20}	3.33	4.33	6.00	**1.00**	2.33	4.00
	Mean	3.27	4.31	4.84	**1.91**	3.24	3.42
	Std	1.73	1.20	1.21	1.12	0.67	0.94
Composition	f_{21}	3.67	3.00	5.67	**2.33**	3.67	2.67
	f_{22}	**1.00**	5.00	6.00	2.00	4.00	3.00
	f_{23}	5.00	6.00	4.00	**1.00**	3.00	2.00
	f_{24}	5.00	6.00	4.00	**1.00**	2.67	2.33
	f_{25}	4.67	6.00	4.33	**1.00**	2.33	2.67
	f_{26}	4.00	5.00	5.33	**2.00**	2.33	2.33
	f_{27}	5.00	6.00	3.33	**1.00**	3.33	2.33
	f_{28}	3.33	3.00	6.00	**2.33**	3.67	2.67
	Mean	3.96	5.00	4.83	**1.58**	3.13	2.50
	Std	1.27	1.22	0.97	0.59	0.60	0.29
Mean		3.76	4.48	4.55	**1.90**	3.20	3.11
Std		1.87	1.42	1.63	**1.20**	1.00	1.14

Heterogeneous Particle Swarm Optimization

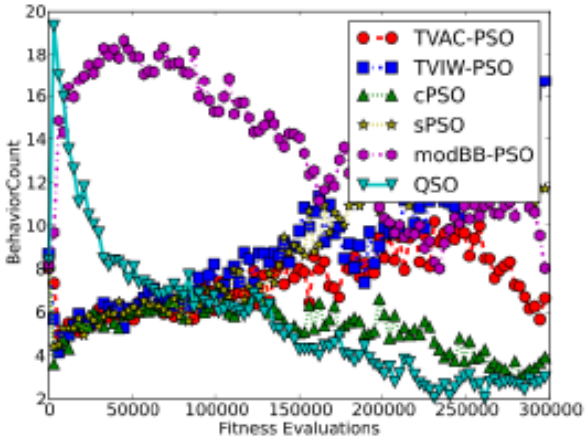
Self-Adaptive HPSO: Frequency-Based Behaviour Profiles



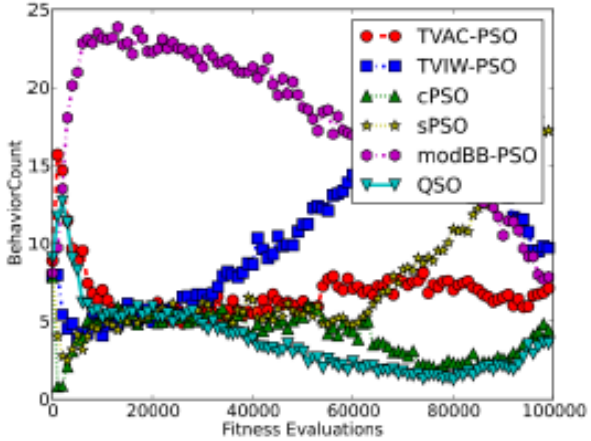
(a) f_4



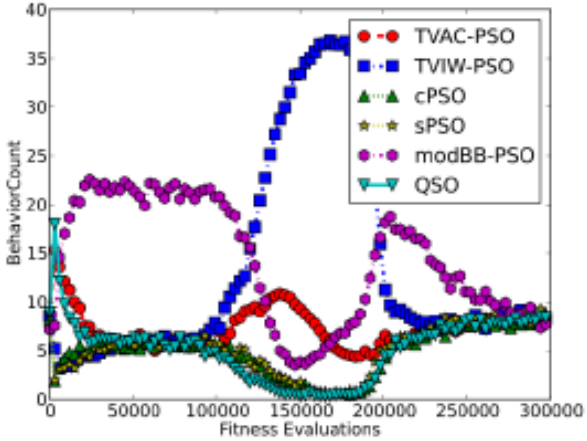
(b) f_{12}



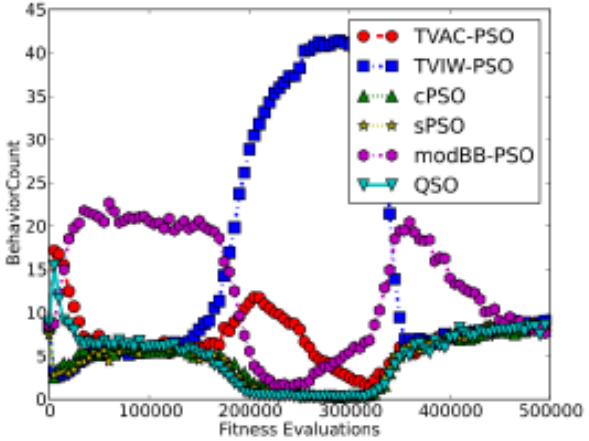
(c) f_{27}



(a) 10D



(b) 30D



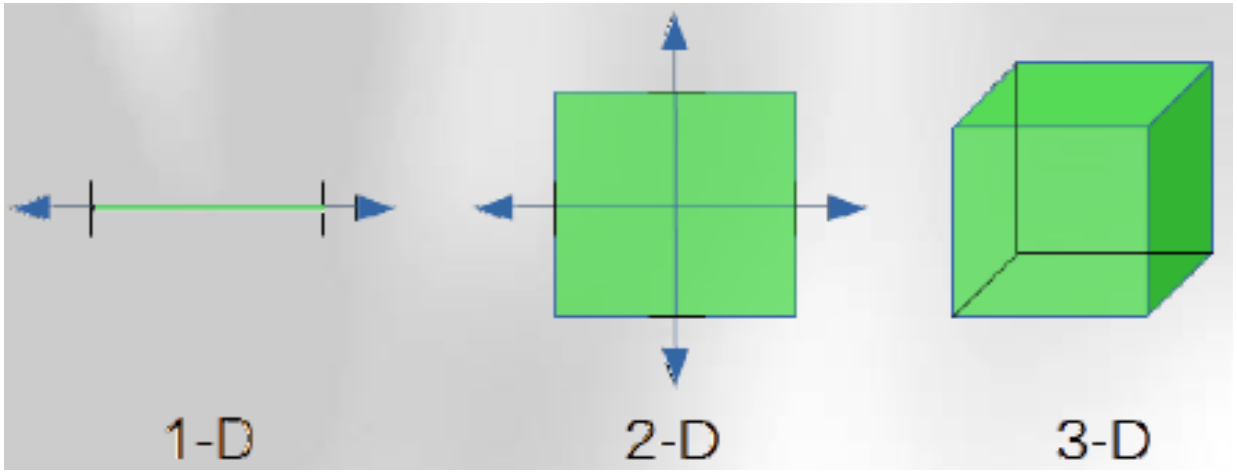
(c) 50D

The following are covered in this part:

- Large-scale optimization problems
- Discrete-valued optimization problems
- Multi-modal optimization
- Dynamic optimization problems
- Constrained optimization problems
- Multi-objective optimization problems
- Many-objective optimization problems

The curse of dimensionality:

- Performance deteriorates with increase in problem dimension



Variable dependencies also becomes a problem

How do we scale particle swarm optimization (PSO) to large scale optimization problems (LSOPs) to efficiently address the probems above?

Do we simply add more particles?

Large Scale Optimization

Cooperative Particle Swarm Optimization[15]

A divide-and-conquer approach to address the curse of dimensionality:

- Each particle is split into K separate parts of smaller dimension
- Each part is then optimized using a separate sub-swarm
- If $K = n_x$, each dimension is optimized by a separate sub-swarm

[15] F van den Bergh, AP Engelbrecht, *A Cooperative Approach to Particle Swarm Optimization*, IEEE Transactions on Evolutionary Computation, 8(3):225–239, 2004

What are the issues with this approach?

- How is the the quality of subswarm particles evaluated?
  - Problem: objecive function is defined for the full n-dimensional problem
  - Solution: Use a context vector to represent a full solution, constructed from the best components from each subswarm

The CPSO-S_K algorithm:

$K_1 = n_x \bmod K$ and $K_2 = K - (n_x \bmod K)$;
Initialize K_1 $\lceil n_x/K \rceil$-dimensional and 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 $f(\mathbf{b}(k, S_k.\mathbf{y}_i)) < f(\mathbf{b}(k, S_k.\hat{\mathbf{y}}))$ **then**
        $S_k.\hat{\mathbf{y}} = S_k.\mathbf{y}_i$;
      end
    end
    Apply velocity and position updates;
  end
until *stopping condition is true*;

Cooperative PSO (CPSO) assumes that variables are independent of one another

What do we do when variable dependencies exist?

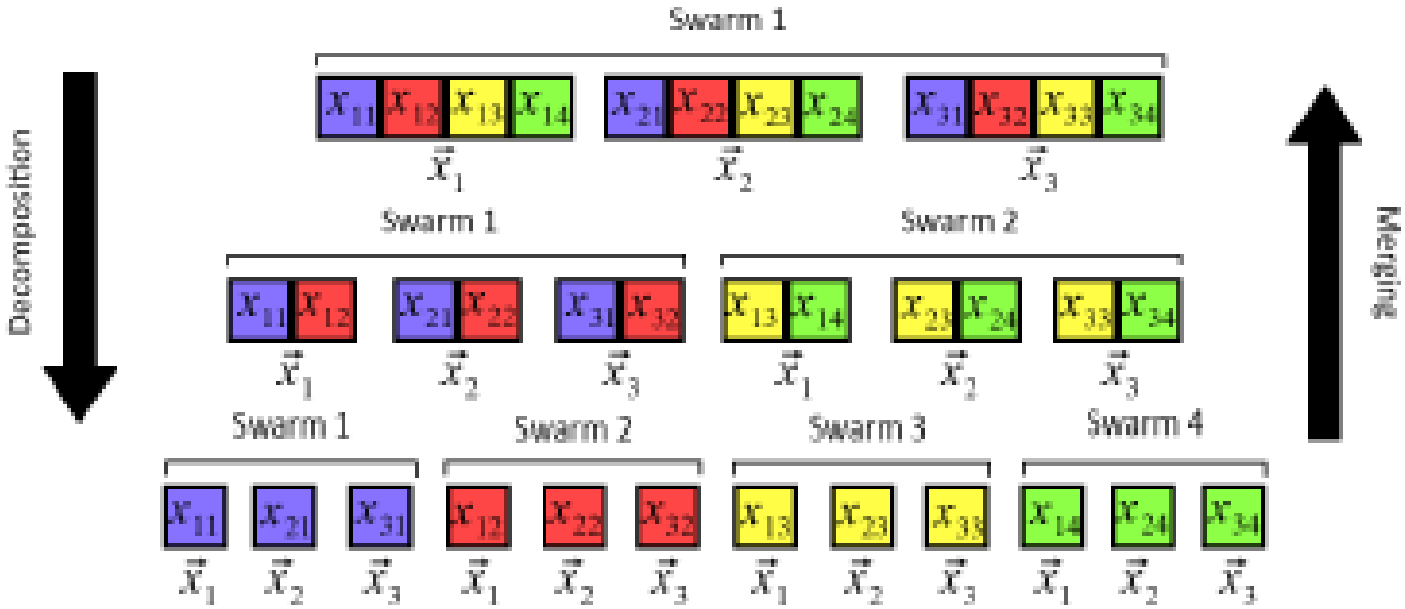
- Increase the dimensionality of the subswarms to optimize more than one decision variable
- Group dependent variables in the same subswarm
- Hybrid CPSO
- Random grouping CPSO

Alternative:

- Decomposition CPSO (DCPSO)
- Merging CPSO (MCPSO)

Large Scale Optimization

Decomposition and Merging CPSO[16]



[16] J Douglas, AP Engelbrecht, BM Ombuki-Berman, *Merging and Decomposition Variants of Cooperative Particle Swarm Optimization: New Algorithms for Large Scale Optimization Problems*, International Conference on Intelligent Systems, Metaheuristics & Swarm Intelligence, 2019

Algorithm 3 Decomposition Algorithm

Initialize an n-dimensional PSO: P
repeat
  for each swarm $j = 1, \ldots, k$ **do**
    for each particle $i = 1, \ldots, s$ **do**
      if $f(\mathbf{b}(j, P_j.\mathbf{x}_i)) < f(\mathbf{b}(j, P_j.\mathbf{y}_i))$ **then**
        $P_j.\mathbf{y}_i = P_j.\mathbf{x}_i$
      if $f(\mathbf{b}(j, P_j.\mathbf{y}_i)) < f(\mathbf{b}(j, P_j.\hat{\mathbf{y}}))$ **then**
        $P_j.\hat{\mathbf{y}} = P_j.\mathbf{y}_i$
    for each particle $i = 1, \ldots, s$ **do**
      Update particle velocity, $P_j.\mathbf{v}_i$ (Equation 2)
      Update particle position, $P_j.\mathbf{x}_i$ (Equation 1)
  if decomposition condition is true **then**
    decompose each P_j into n_r sub-swarms
until stopping criterion is met

Algorithm 4 Merging Algorithm

Initialize n one-dimensional PSOs: $P_j, j = 1, \ldots, n$
repeat
  for each swarm $j = 1, \ldots, k$ **do**
    for each particle $i = 1, \ldots, s$ **do**
      if $f(\mathbf{b}(j, P_j.\mathbf{x}_i)) < f(\mathbf{b}(j, P_j.\mathbf{y}_i))$ **then**
        $P_j.\mathbf{y}_i = P_j.\mathbf{x}_i$
      if $f(\mathbf{b}(j, P_j.\mathbf{y}_i)) < f(\mathbf{b}(j, P_j.\hat{\mathbf{y}}))$ **then**
        $P_j.\hat{\mathbf{y}} = P_j.\mathbf{y}_i$
    for each particle $i = 1, \ldots, s$ **do**
      Update particle velocity, $P_j.\mathbf{v}_i$ (Equation 2)
      Update particle position, $P_j.\mathbf{x}_i$ (Equation 1)
  if merging condition is true **then**
    merge every n_r sub-swarms
until stopping criterion is met

Large Scale Optimization

Decomposition and Merging CPSO: Results

- $n_x \in \{30, 500, 1000\}$
- Number of subswarms: 6
- Particles per subswarm: 10
- $w = 0.72$
- $c_1 = c_2 = 1.49$
- 30 independent runs
- 95% confidence interval using Mann-Whitney U tests

Test Function, f(x)	Domain	Class
Absolute Value, f_1	[-100 .. 100]	Unimodal Separable
Ackley, f_2	[-30 .. 30]	Multimodal Non-separable Rotated
Egg Holder, f_3	[-512 .. 512]	Multimodal Non-separable
Elliptic, f_4	[-100 .. 100]	Unimodal Separable Rotated
Generalized Griewank, f_5	[-600 .. 600]	Multimodal Non-separable Rotated
Hyperellipsoid, f_6	[-5.12 .. 5.12]	Unimodal Separable
Michalewicz, f_7	[0 .. π]	Multimodal Separable
Norwegian, f_8	[-1.1 .. 1.1]	Multimodal Non-separable
Quadric, f_9	[-100 .. 100]	Unimodal Non-separable
Quartic, f_{10}	[-1.28 .. 1.28]	Unimodal Separable
Generalized Rastrigin, f_{11}	[-5.12 .. 5.12]	Multimodal Separable Rotated
Generalized Rosenbrock, f_{12}	[-30 .. 30]	Multimodal Non-separable Rotated
Salomon, f_{13}	[-100 .. 100]	Multimodal Non-separable
Schaffer 6, f_{14}	[-100 .. 100]	Multimodal Non-separable
Schwefel 1.2, f_{15}	[-100 .. 100]	Unimodal Non-separable Rotated
Schwefel 2.21, f_{16}	[-100 .. 100]	Unimodal Separable
Spherical, f_{17}	[-5.12 .. 5.12]	Unimodal Separable
Step, f_{18}	[-100 .. 100]	Multimodal Separable
Vincent, f_{19}	[0.25 .. 10]	Multimodal Separable
Weierstrass, f_{20}	[-0.5 .. 0.5]	Multimodal Separable

Large Scale Optimization

Decomposition and Merging CPSO: Results

n_x	Problem Class	CPSO Algorithm				
		CPSO-S_k	**CPSO-H_k**	**CCPSO**	**DCPSO**	**MCPSO**
50	U	2.0	3.33	3.67	2.33	3.33
	M	3.0	3.67	2.67	1.33	2.33
	S	2.0	2.5	3.5	2.5	2.0
	NS	3.0	4.0	3.0	1.5	3.25
	Avg	2.5	3.25	3.25	2.0	2.63
500	U	2.67	4.67	1.67	2.0	2.0
	M	2.67	5.0	3.67	1.67	1.67
	S	3.5	5.0	3.5	2.0	1.0
	NS	3.75	4.75	1,75	1.67	2.25
	Avg	3.625	4.88	2.63	1.88	1.63
1000	U	4.0	4.33	1.67	2.0	2.0
	M	3.67	4.33	2.33	1.0	2.0
	S	3.0	4.0	2.5	1.5	1.0
	NS	4.0	4.5	1.75	1.5	2.5
	Avg	3.5	4.25	2.13	1.5	1.75

While the CPSO algorithms helped to make PSO more scalable, we need to gain a better understanding of why PSO does not scale well

Compare PSO's performance on low and high dimensional problems:

- $n_x \in \{10, 1000\}$
- $n_s = 30$
- Global best PSO
- Only update personal best and global best if new positions are better and feasible
- $w = 0.7298, c_1 = c_2 = 1.49618$
- $n_t = 5000$

Large Scale Optimization

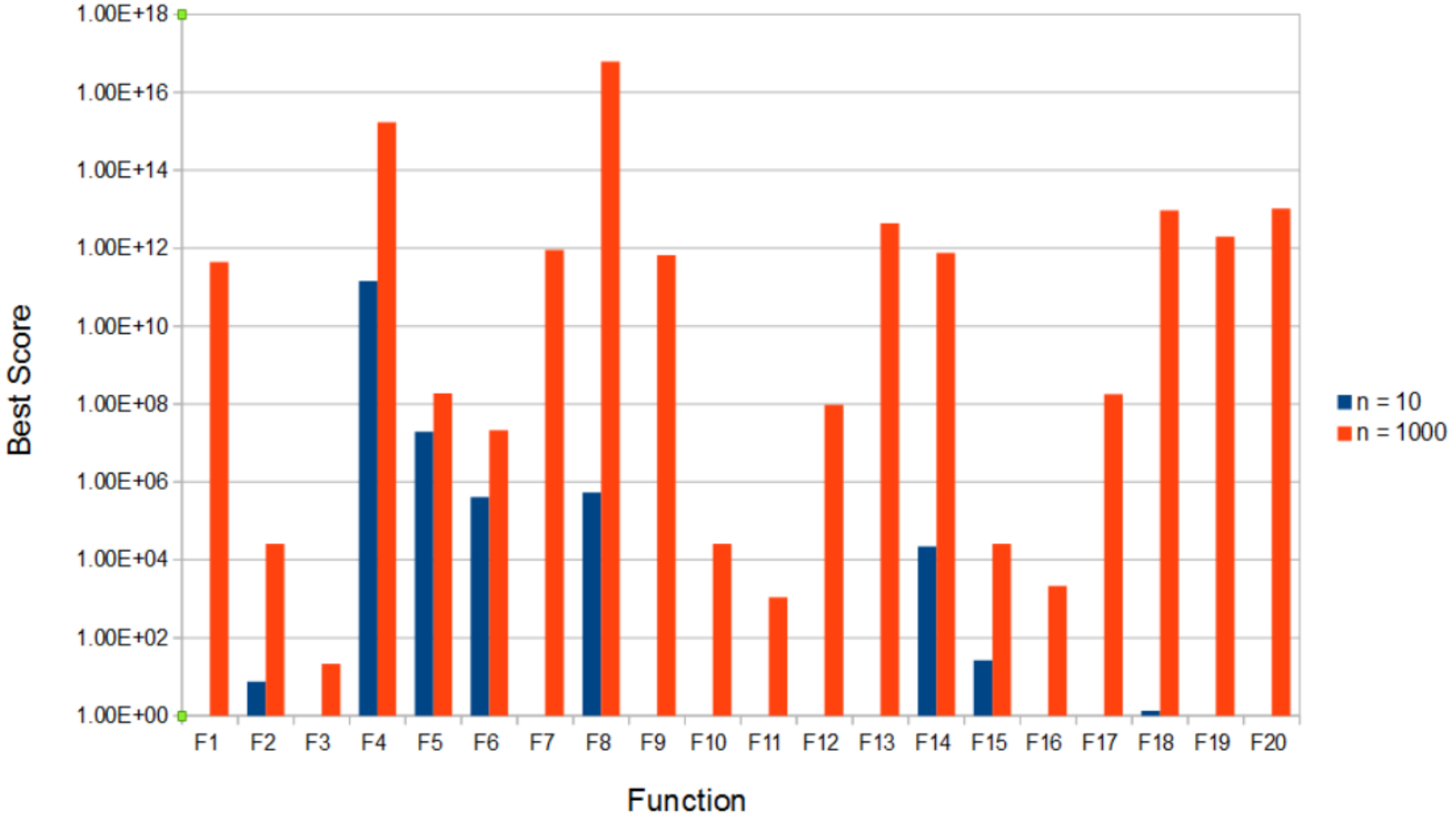
Understanding Lack of Scalability (cont)

- CEC 2020 benchmark suite for large scale optimization
- 20 benchmark functions
- Minimization problems with optimum of 0
- 30 independent runs of each algorithm configuration on each function
- Friedman test to detect if significant differences exist
- If so, then pairwise Mann-Whitney U tests, with confidence 0.05
- Score reflects performance over all functions

Large Scale Optimization

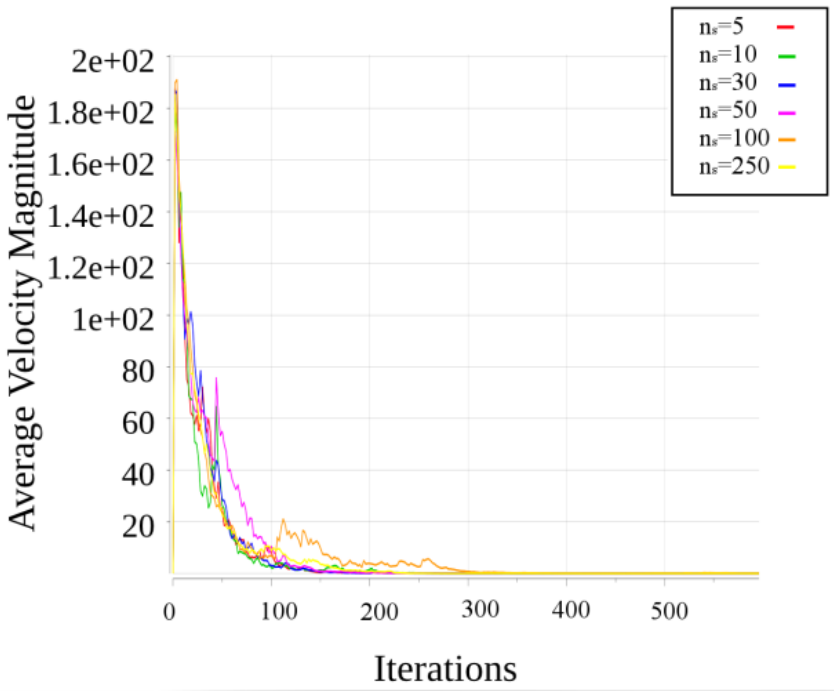
Understanding Lack of Scalability (cont)

Logarithmic plot of best score achieved for each function

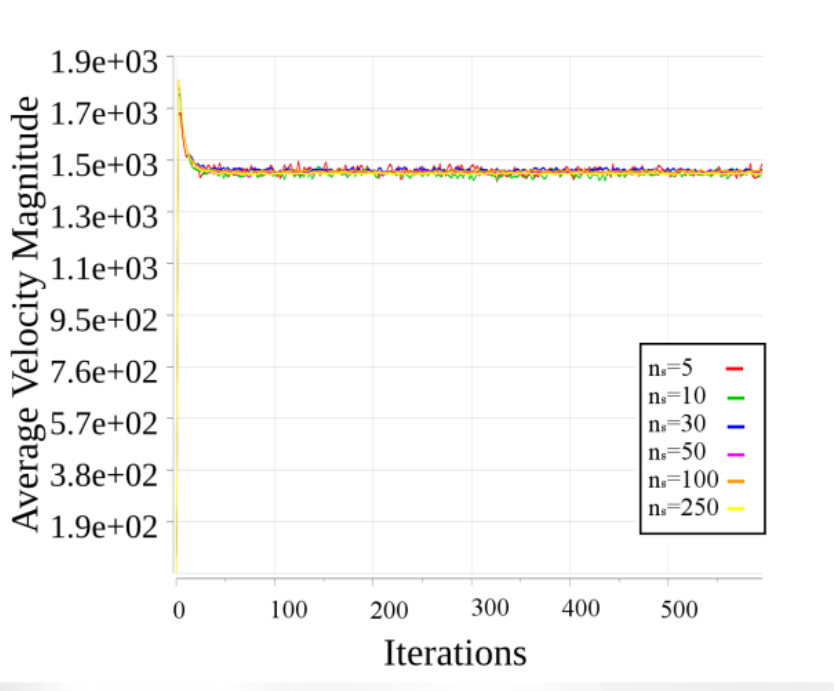


Average particle velocity on F7 for varying swarm sizes

$n_x = 10$

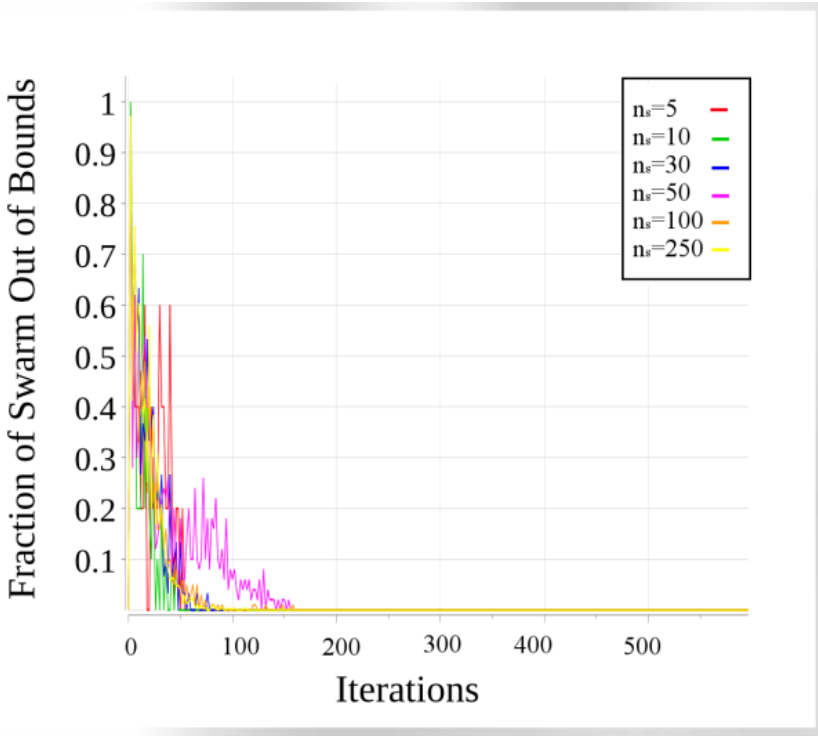


$n_x = 1000$

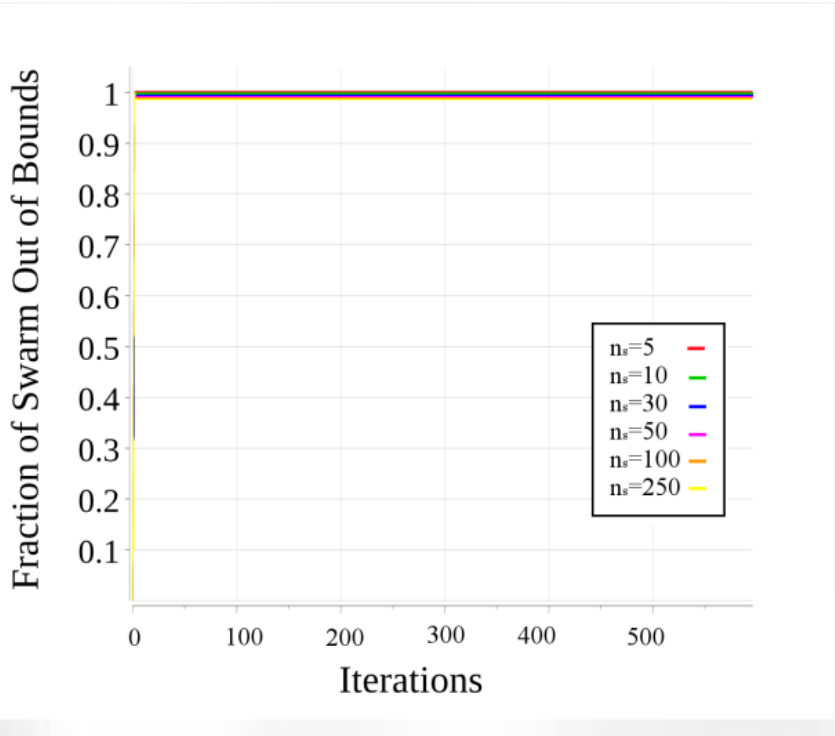


Fraction of swarm outside of bounds for F7 for varying swarm sizes

$n_x = 10$



$n_x = 1000$



The question then is if the velocity explosion and subsequent roaming behavior can be mitigated?

Potential solutions:

- We have already seen decomposition-based approaches
- Velocity clamping
- Restricting particle variance
- Stochastic scaling
- Initialization schemes

Clamping per dimension

$$v_{i,j}^{t+1} = \begin{cases} v_{i,j}^{t+1} & \text{if } -v_{max,j} \leq v_{i,j}^{t+1} \leq v_{max,j} \\ v_{max,j} & \text{if } v_{max,j} < v_{i,j}^{t+1} \\ -v_{max,j} & \text{if } v_{i,j}^{t+1} < -v_{max,j} \end{cases}$$

v_{max} is a fraction of the search space:

$$v_{max,j} = \delta(U_j - L_j)$$

where $\delta \in (0, 1)$ and bounds in j-th dimension are $[L_j, U_j]$

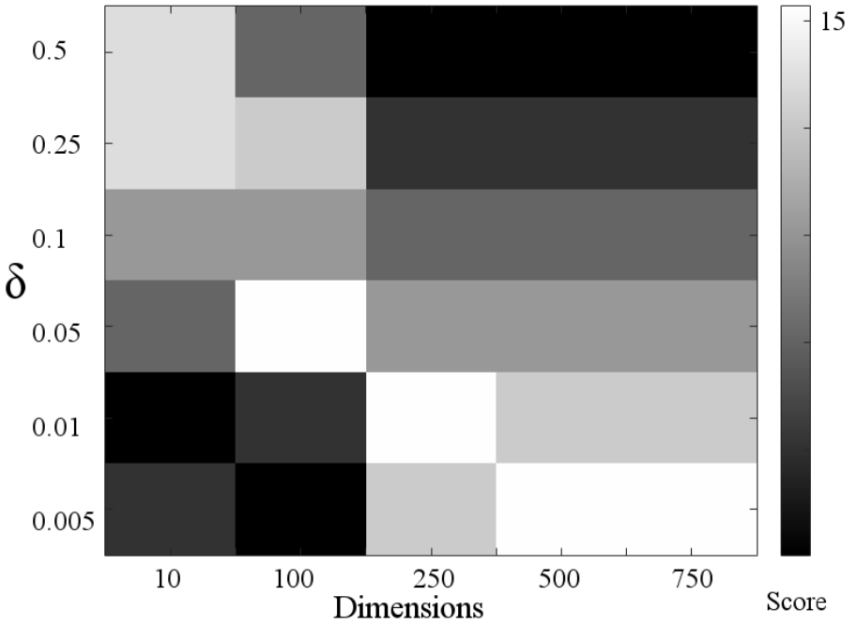
[17] ET Oldewage, AP Engelbrecht, CW Cleghorn, *The Merits of Velocity Clamping Particle Swarm Optimisation in High Dimensional Spaces*, IEEE Swarm Intelligence Symposium, 2017

Magnitude/normalized clamping

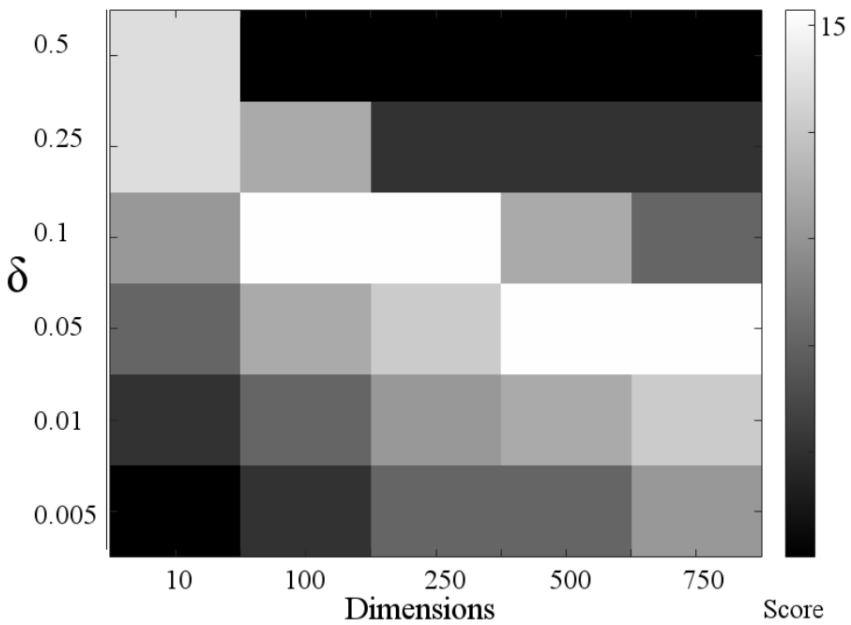
$$\mathbf{v}_i^{t+1} = \begin{cases} \mathbf{v}_i^{t+1} & \text{if } \|\mathbf{v}_i^{t+1}\| \leq v_{max} \\ \frac{v_{max}}{\|\mathbf{v}_i^{t+1}\|}\mathbf{v}_i^{t+1} & \text{if } \|\mathbf{v}_i^{t+1}\| > v_{max} \end{cases}$$

$$v_{max} = \delta \sqrt{\sum_{j=1}^{n} (U_j - L_j)^2} = \delta \|\mathbf{U} - \mathbf{L}\|$$

Dimension clamping

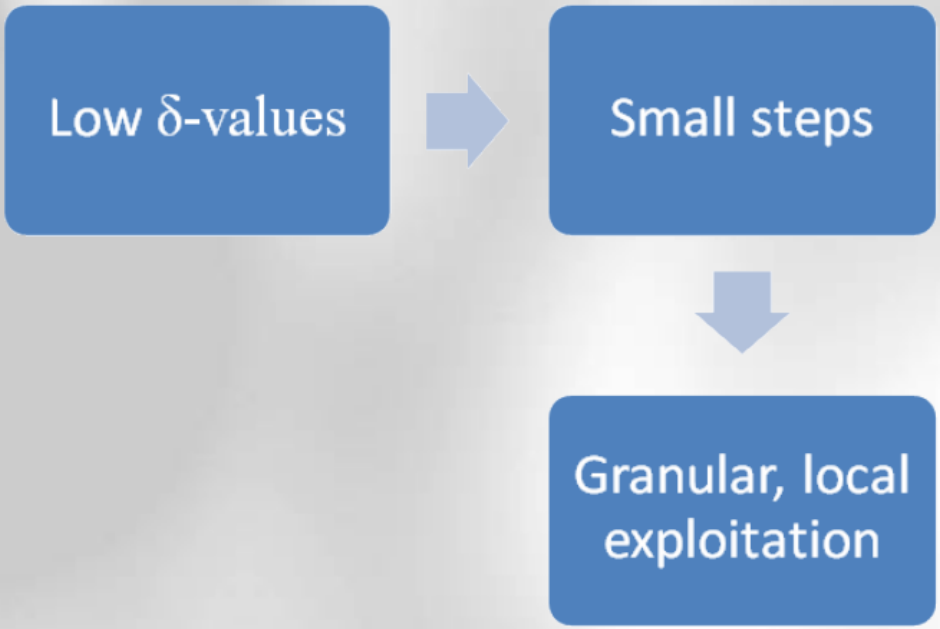


Magnitude clamping

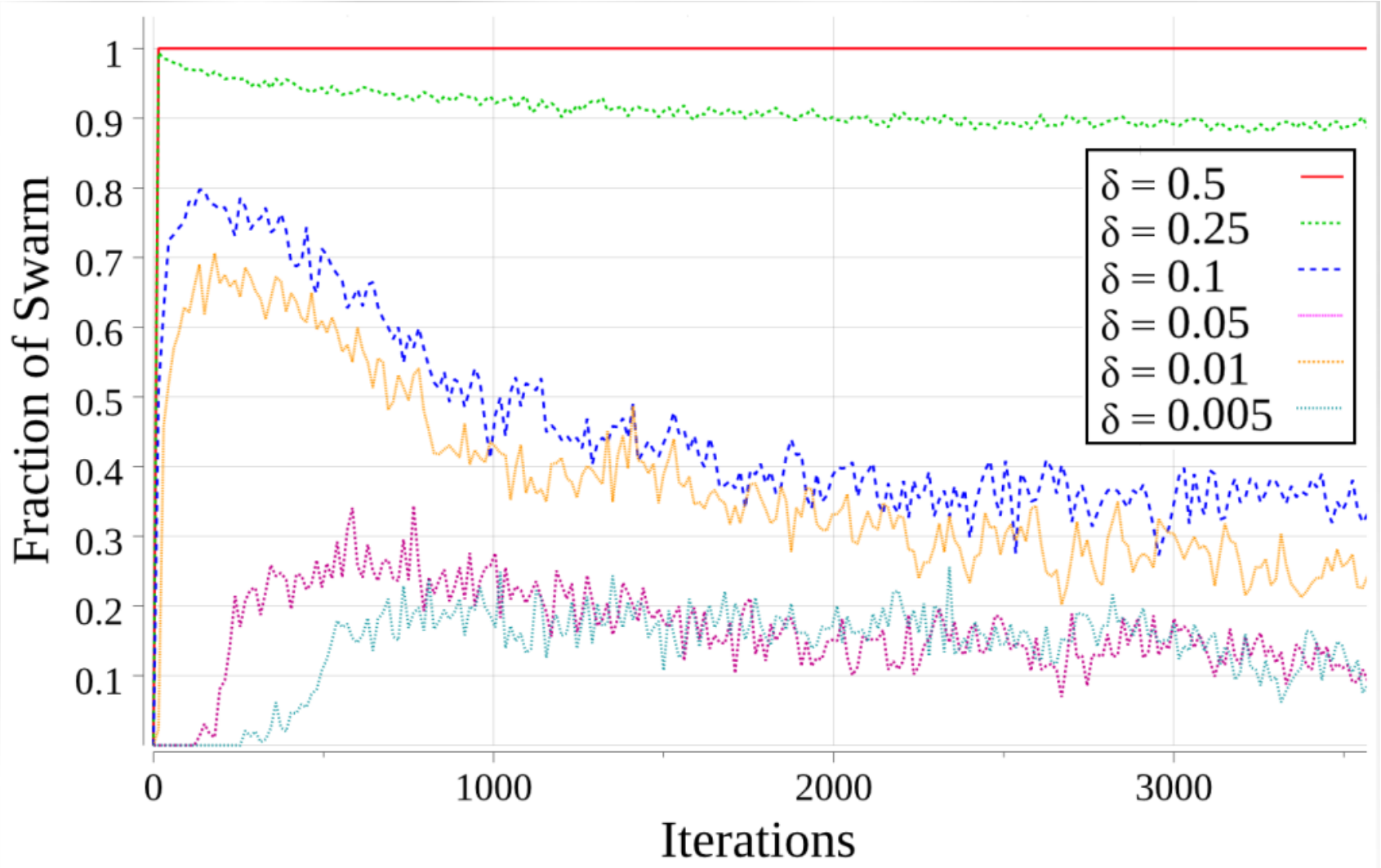


- Little difference in the two clamping approaches for low dimensions
- As dimenions increases, clamping per dimension performs better

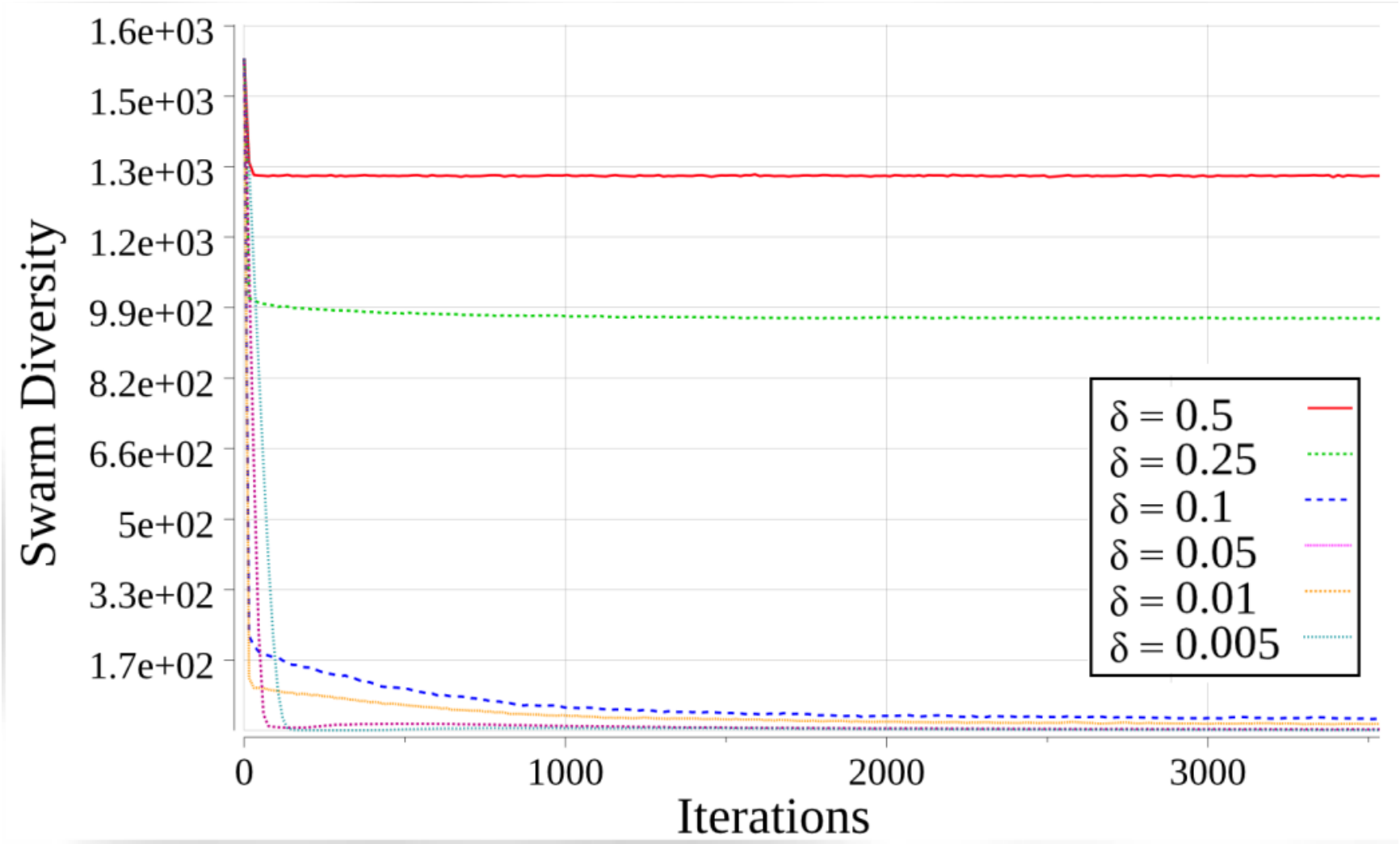
Observation: Optimal δ decreases as dimensionality increases



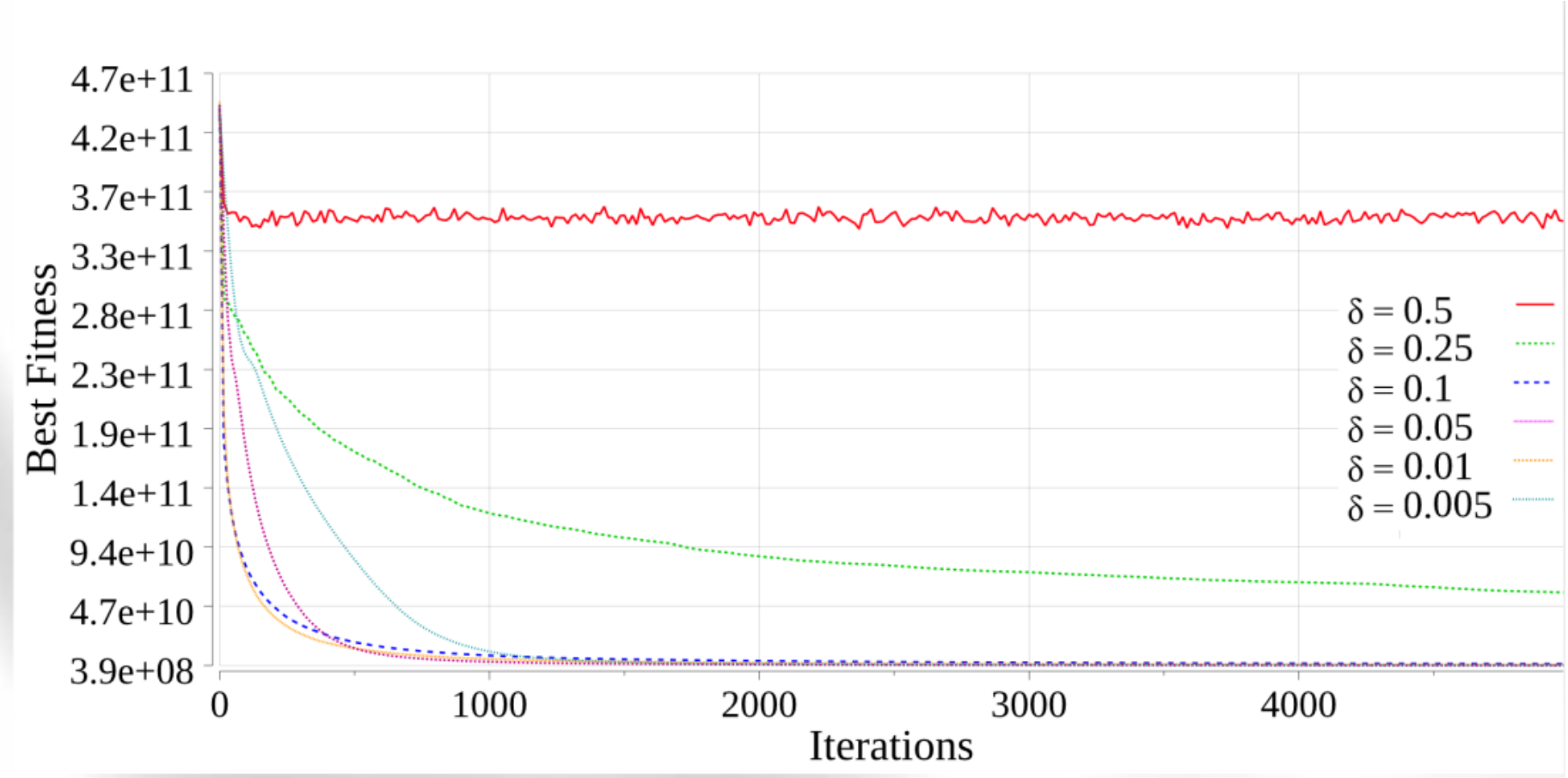
Effect on roaming behavior: Clamp per dimension for F9, $n_x = 750$



Effect on swarm diversity: Clamp per dimension for F9, $n_x = 750$



Effect on solution quality: Clamp per dimension for F9, $n_x = 750$



Summary of findings:

- Clamp more as dimensionality increases
- Clamp per dimension as dimensionality increases
- Particle roaming reduced for strong clamping, but not eliminated
- Solution quality improves for strong clamping, but stil premature stagnation

Large Scale Optimization

Understanding Lack of Scalability: Restricting Particle Variance[18]

Particle variance refers to the deviation in a particle's positions:

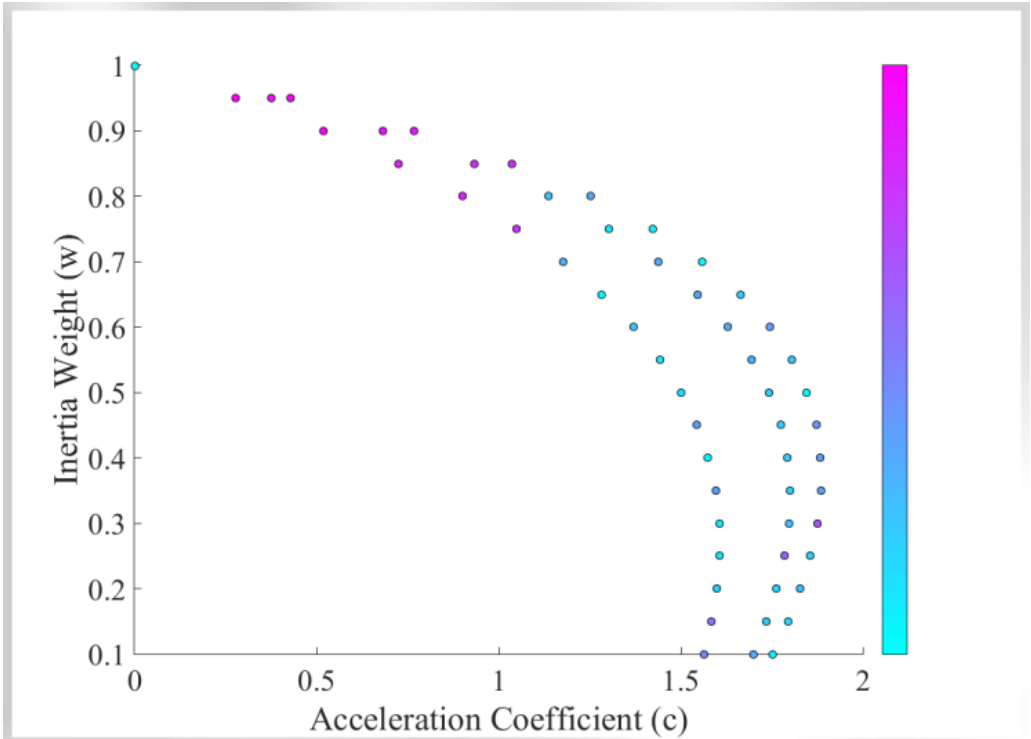
- can we restrict the variance (deviation) in particle movement to facilitate exploitation?
- resulting in more smooth search trajectories.
- will this reduce velocity explosion?
- how to set w and c_1, c_2 to restrict particle movement in high dimensional spaces?

Deviation in particle's positions:

$$\sigma = \frac{1}{2}\sqrt{\frac{c(w+1)}{c(5w-7)-12w^2+12}}|\hat{y}-y|$$
$$= V_c|\hat{y}-y|$$

[18] ET Oldewage, AP Engelbrecht, CW Cleghorn, *Movement Patterns of a Particle Swarm in High Dimensions*, Information Sciences, 512:1043–1062, 2020

Best configurations have high inertia weight and low acceleration coefficients



Performance of swarm configuration produced by restricting standard deviation of positions (pink is better) ($c = c_1 = c_2$)

The findings:

- Restricting particle variance reduces the velocity explosion
- A high inertia weights and low acceleration coefficients perform well in high dimensions due to the regularizing influence of the inertia weight
- High inertia weights and low acceleration coefficients bring about smooth particle trajectories

Large Scale Optimization

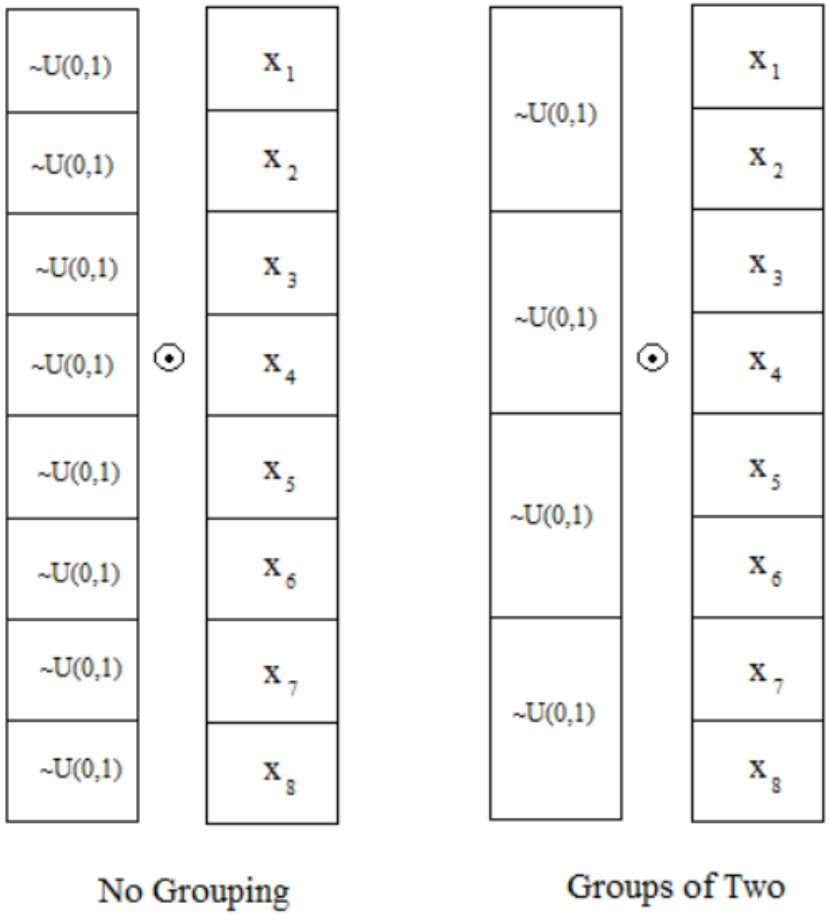
Understanding Lack of Scalability: Stochastic Scaling[19]

Stochastic scaling refers to the amount of randomness in the velocity updates:

- If r_1 and r_2 are scalars, every position that can be reached by the swarm is a linear combination of the particles' initial positions $\rightarrow$ less exploration
- If they are vectors, more exploration, and exploration outside of initial subspace
- Group-based scaling: something between scalar and stochastic scaling

[19] E Oldewage, AP Engelbrecht, CW Cleghorn, *Degrees of Stochasticity in Particle Swarm Optimization*, Swarm Intelligence, 13(3-4):193-215, 2019

Variable grouping/coupling:

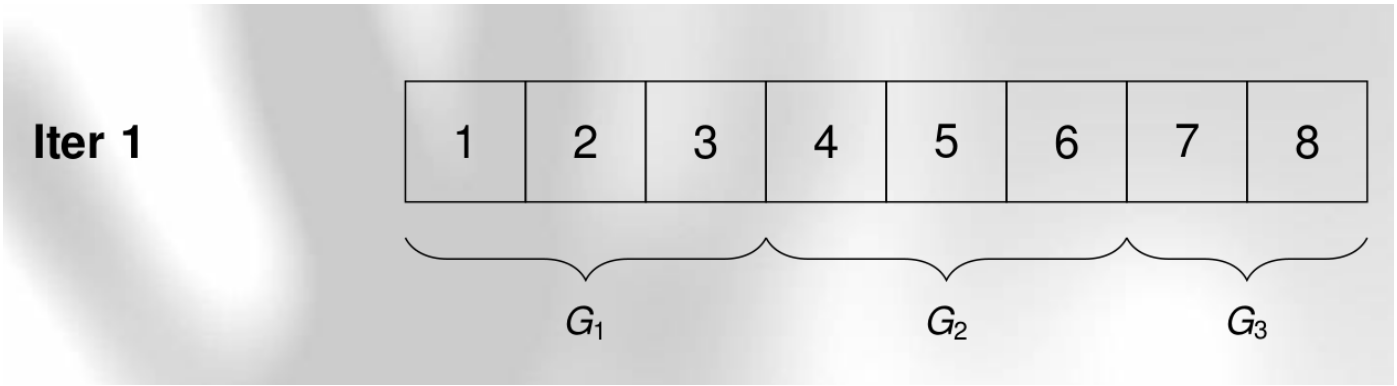


- Divides decision variables into q groups
- $\mathbf{r}_1, \mathbf{r}_2$ are q-dimensional vectors
- Apply the same stochastic scalar to all decision variables in a group
- Reduce degrees of freedom by introducing coupling

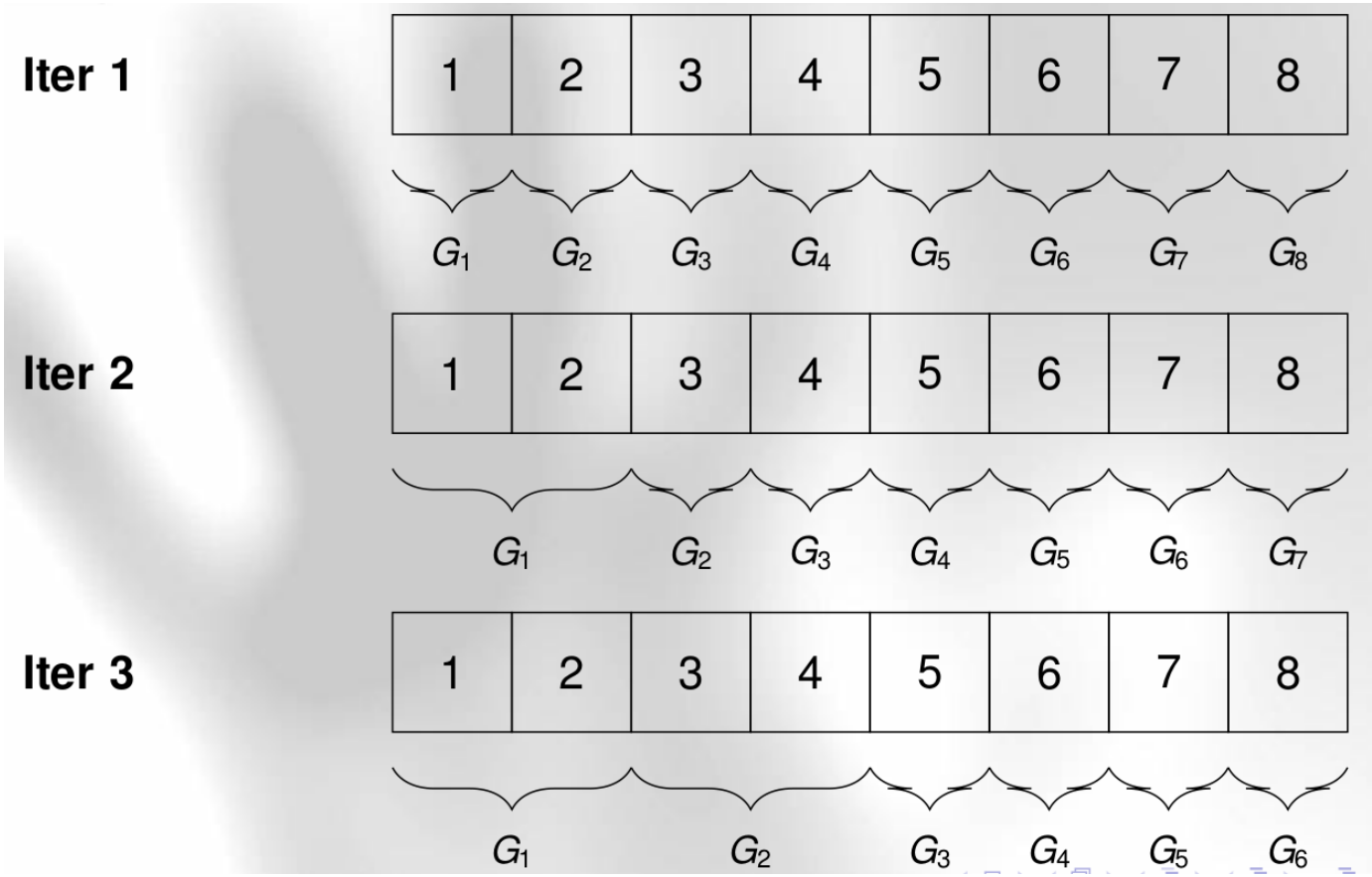
Three approaches:

- Fixed number of groups
- Linearly decreasing number of groups
- Linearly increasing number of groups

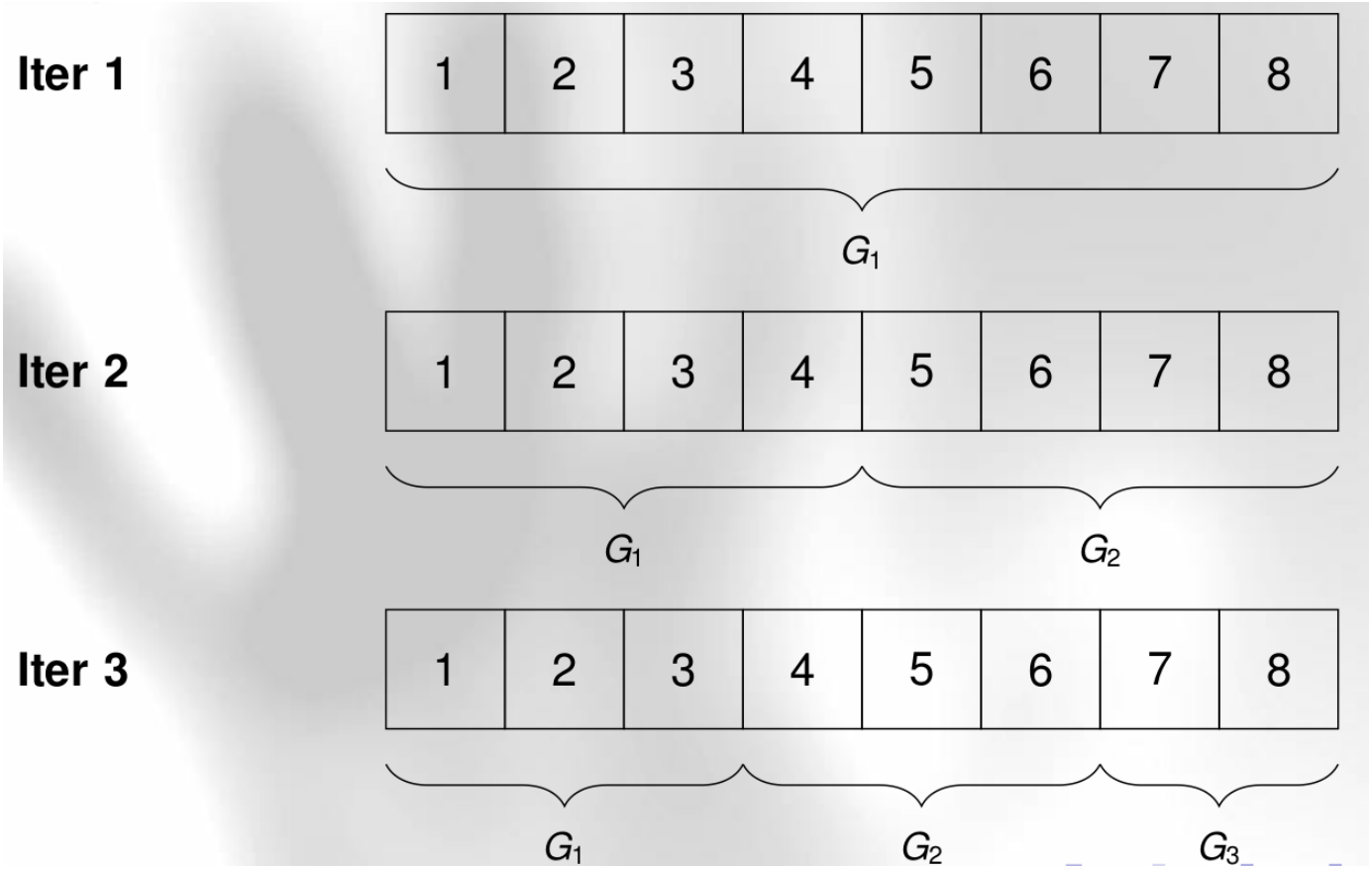
Fixed number of groups:



Decreasing number of groups:



Increasing number of groups:



All of the PSOs with group scaling performed significantly better than canonical PSO on all the benchmark functions.

Table: Swarms with Grouping-Based Stochastic Scaling vs Simple Swarm

	$>$ Simple	$=$	$<$ Simple
Static Fixed	20	0	0
Static Decreasing	20	0	0
Static Increasing	20	0	0

Restricting swarm movement forces exploitation of the subspace to which it is restricted rather than attempting to explore a huge search space

Increasing number of groups strategy performed significantly better than the other two strategies.

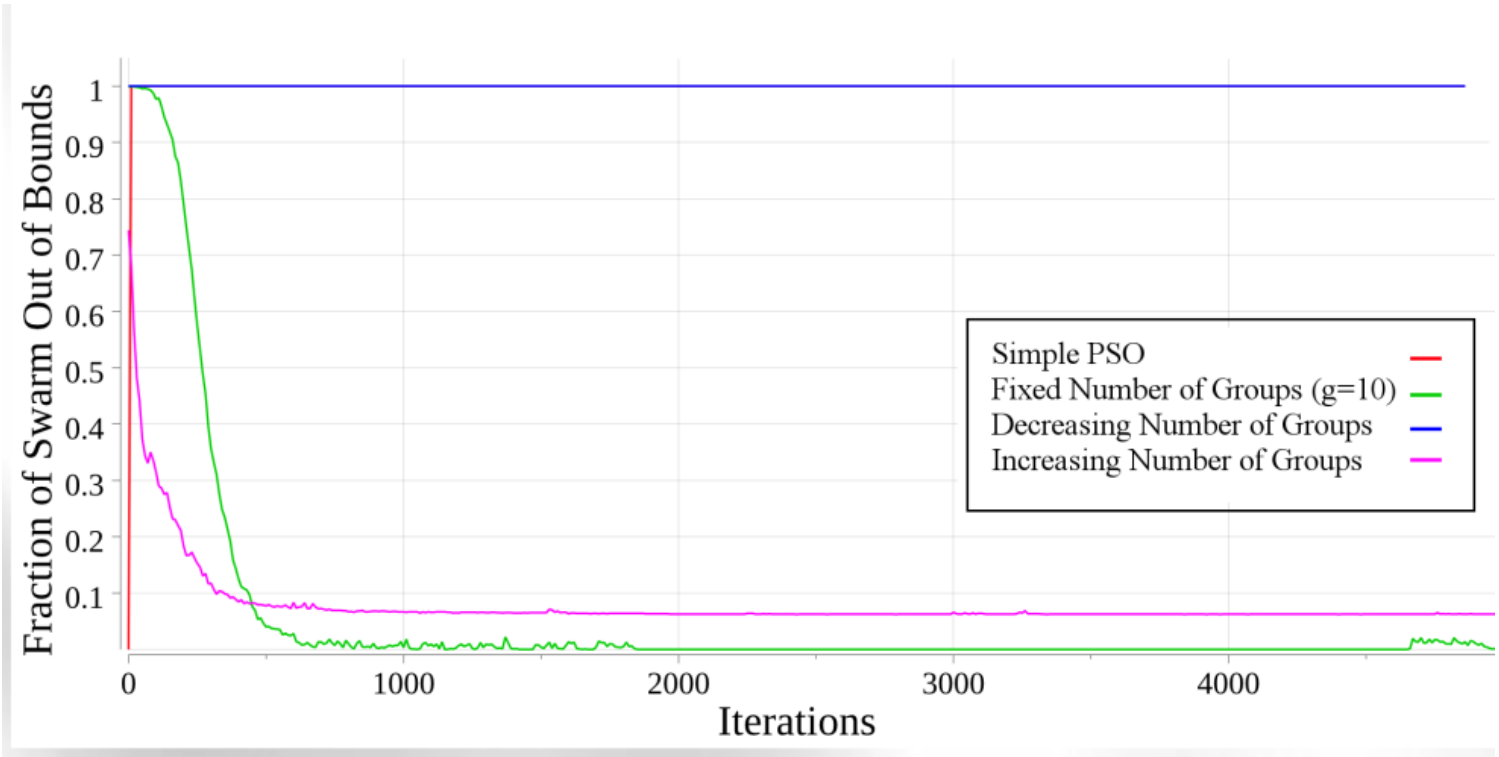
Table: Comparison of Increasing Group Number Strategy with Fixed (g = 10) and Decreasing Group Number Strategies

	$>$ Increasing	$=$	$<$ Increasing
Fixed	0	3	17
Decreasing	0	0	20

Why do increasing and fixed group numbers do better than decreasing group numbers?

- Both the increasing and fixed group number strategies restrict particle movement in the first few iterations

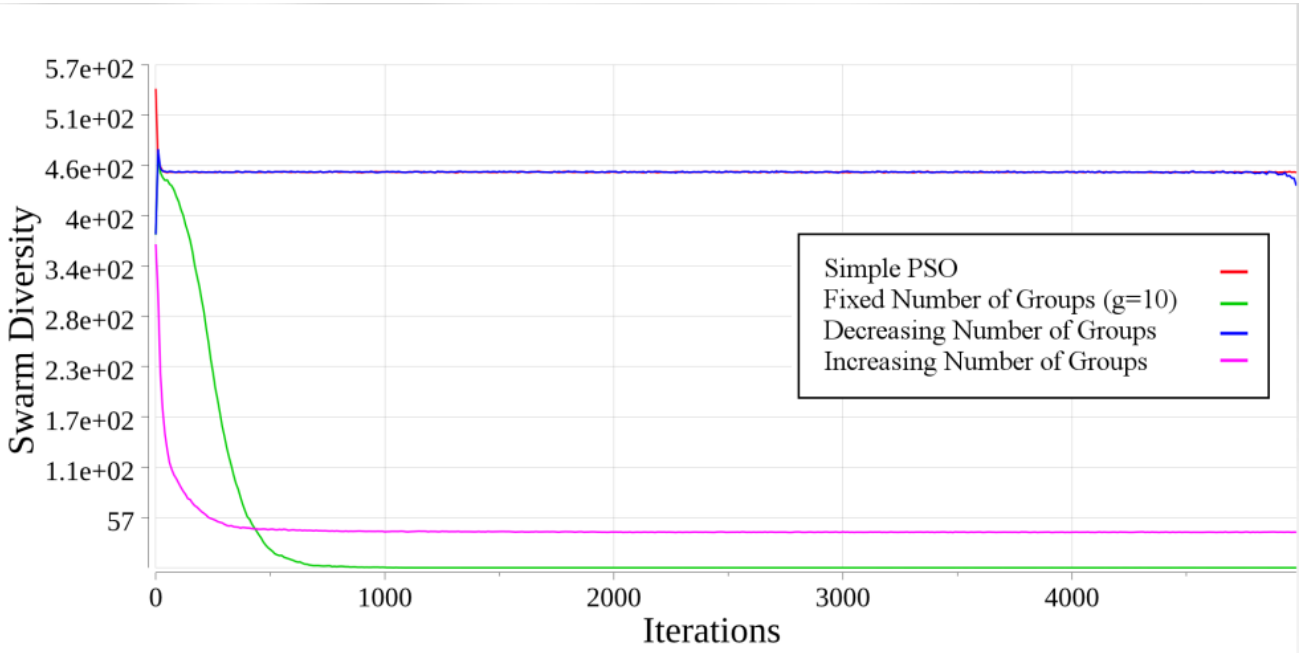
Roaming for Ackley function, $n_x = 1000, n_t = 2000$



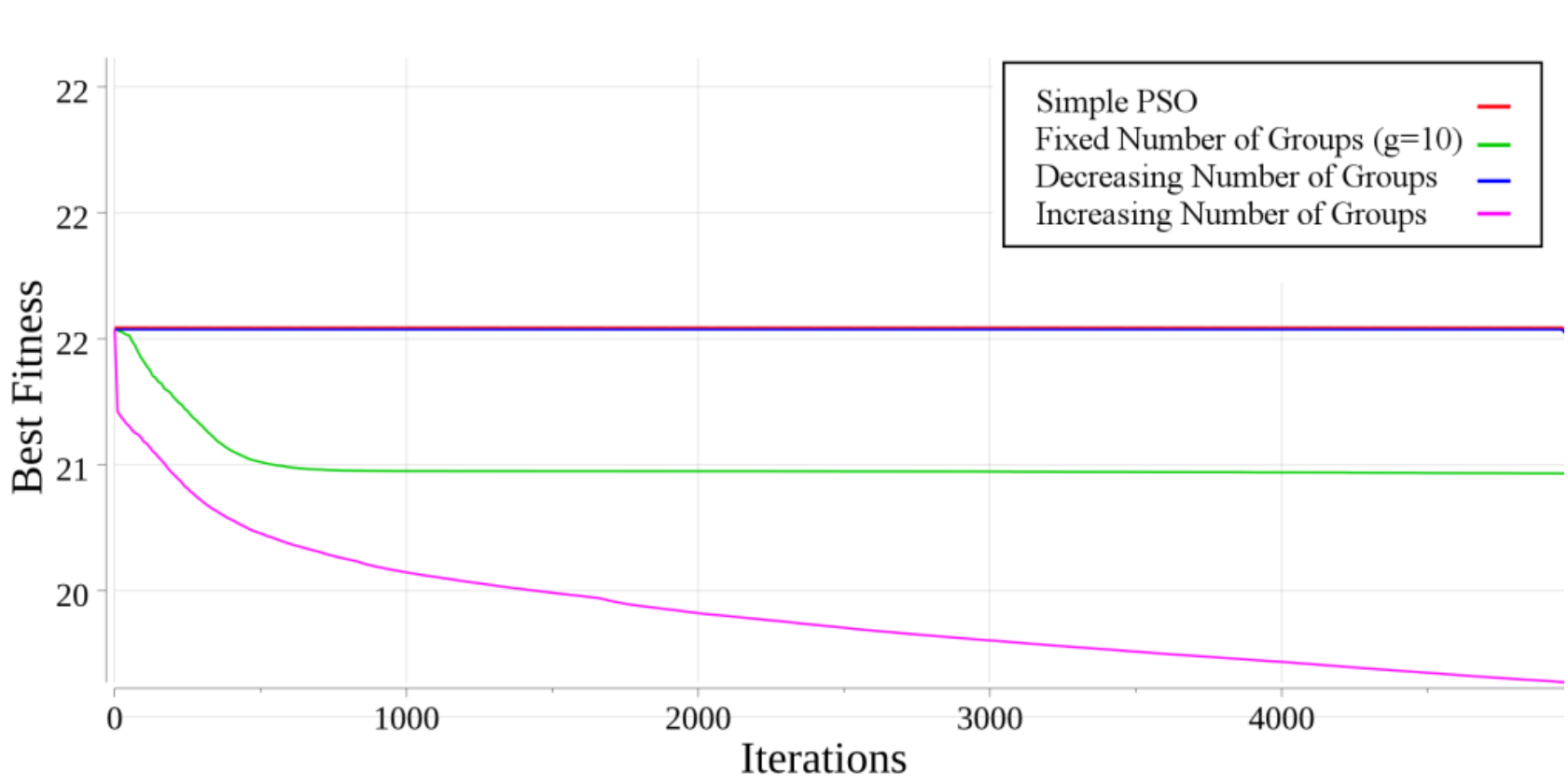
Why do fixed group numbers do worse than increasing group numbers?

- Fixed group numbers may prevent particles from being able to escape an unfavourable initial subspace
- Increasing group numbers will allow particles to escape such subspaces

Solution quality for Ackley function, $n_x = 1000, n_t = 2000$



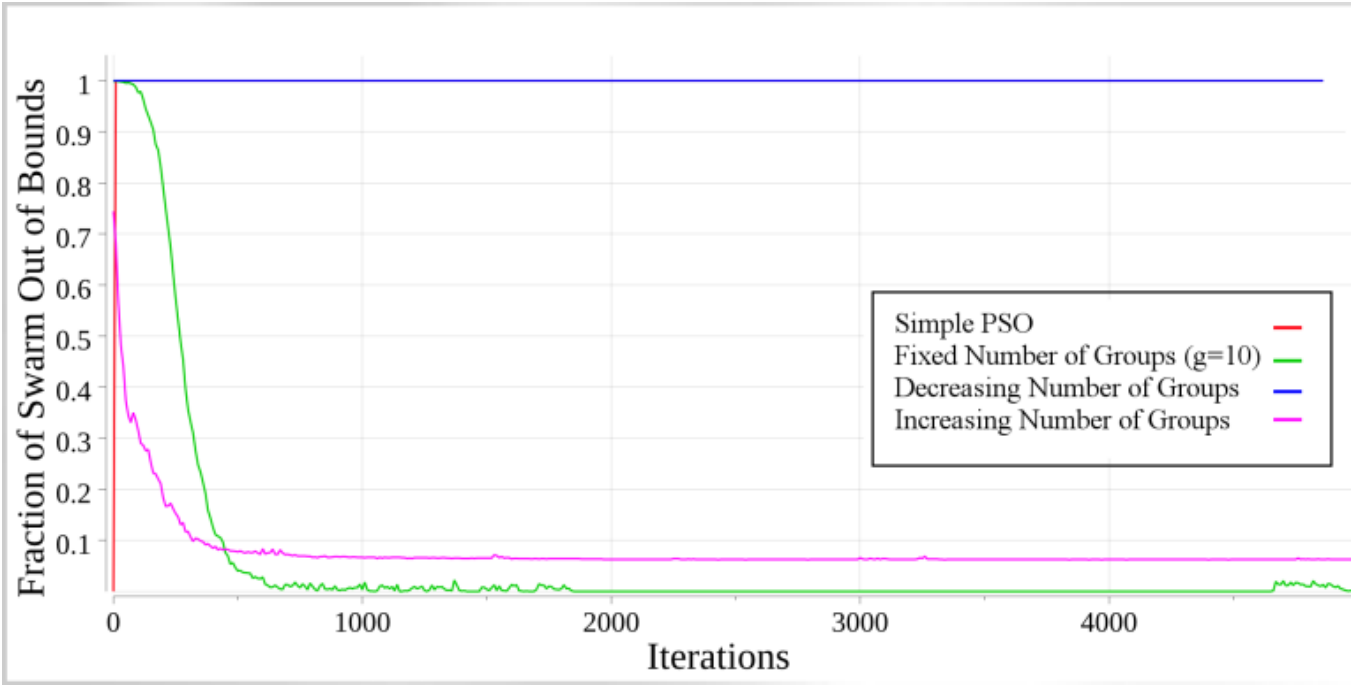
Swarm diversity for Ackley function, $n_x = 1000, n_t = 2000$



Why does increasing the group numbers do well?

- The initial velocity explosion is prevented and the swarm is allowed more degrees of freedom to prevent premature convergence

Roaming for F3 function, $n_x = 1000, n_t = 5000$



Summary of observations:

- Restricted swarm movement by introducing dependencies between dimensions is beneficial
- All group-based scaling strategies performed better than a simple PSO
- Increasing the number of groups was the most successful strategy
- Initially restricting swarm movement prevents velocity explosion and reduces roaming

For low dimensional search spaces, particles' initial position should uniformly cover the search space

For high dimensional search spaces:

- the region of the search space that can be covered by particles' initial positions is very small; search space grows exponentially with dimensionality
- even for uniformly distributed particles, much of the search space remains unseen due to its sheer size
- Helwig and Wanka have shown that, in high dimensional spaces, particles that are initialised uniform randomly are located arbitrarily close to the boundary with overwhelming probability
- such particles are likely to leave the search space in the next iteration
- uniform random initialisation may thus exacerbate particle roaming behaviour in high dimensional spaces

Idea to explore:

- in high dimensions, it is more effective to initialise the particles within a small area of the search space rather than attempting to spread the particles as evenly as possible
- restrict swarm movement by initializing particles in a subspace
- focus on searching locally, rather than trying to explore the whole search space

Subspace initialization approach:

- Uses a "seed set", which is simply a set of u, randomly generated, n-dimensional orthogonal unit vectors
- Randomly choose u linearly independent vectors in search space
- Orthogonalize these initial vectors with the Modified Gram-Schmidt method
- The initial set of linearly independent vectors are generated randomly
- This generates a random subspace within search space
- Initialize all particles in the random subspace

Hypothesis:

- As u is increased, more of the search space can be reached
- As u is decreased, the particles must focus on local searching

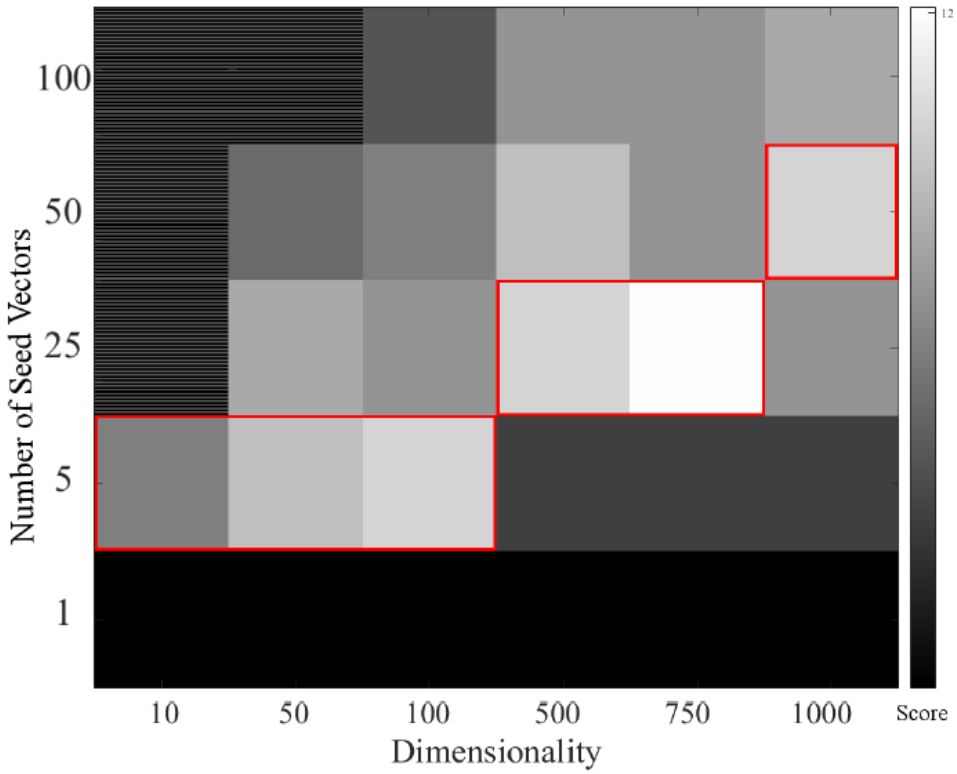
Compared to:

- Uniform Random Initialization
- Sobol Sequence Initialization
- Non-linear Simplex Method
- Centroidal Voronoi Tesselations

Questions:

- How does the optimal seed set size change with dimensionality?
- How does the subspace strategy compare to the others?

Scores of different seed set sizes across dimensionality



Larger seed sets performed better as dimensionality increased

Comparison with subspace-based PSO on $n_x = 10$

	$>$ Subspace u = 5	$=$	$<$ Subspace u = 5
Uniform Random	2	18	0
Sobol Sequence	2	18	0
NSM	0	18	2
CVT	2	18	0

Subspace PSO generally not helpful in low dimensions

But as dimensionality increases...

Comparison with subspace-based PSO on $n_x = 500$

	$>$ Subspace u=25	$=$	$<$ Subspace u=25
Uniform Random	0	0	20
Sobol Sequence	0	0	20
NSM	0	0	20
CVT	7	12	1

Comparison with subspace-based PSO on $n_x = 1000$

	$>$ Subspace u = 50	$=$	$<$ Subspace u = 50
Uniform Random	0	14	6
Sobol Sequence	0	0	20
NSM	0	0	20
CVT	4	16	0

Initialization techniques that maximize initial spread perform badly:

- Attempting to explore the entire search space is not feasible
- Local searching is more beneficial

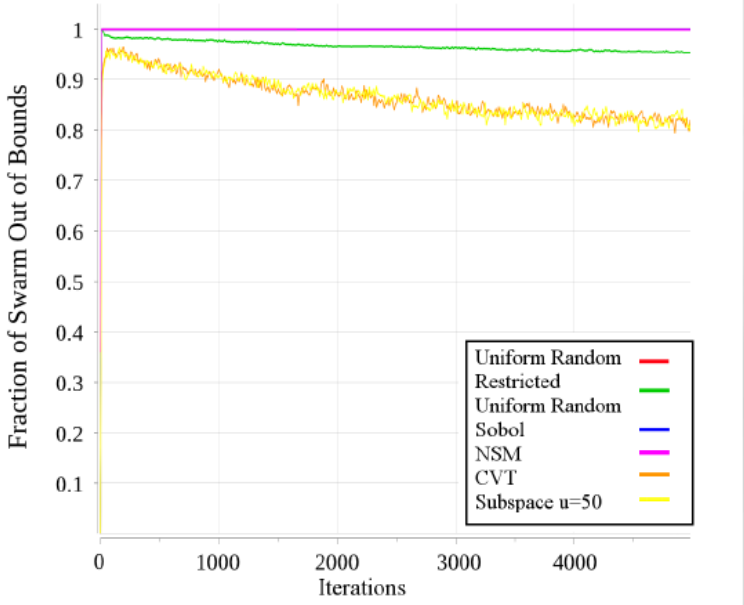
Subspace initialization technique caused particle positions to be small in most dimensions

- Low initial swarm diversity
- Better exploitation/fine-grained exploration

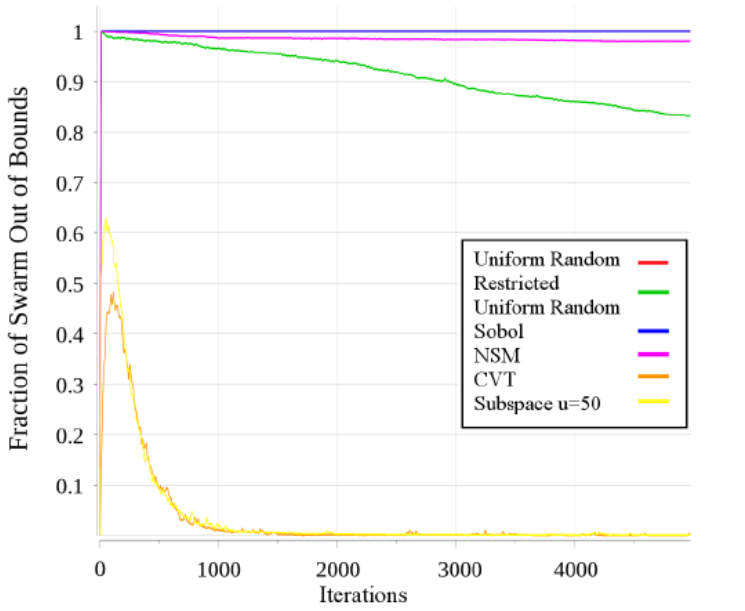
Although the global minimum may not be found, it may be better to compromise by looking for good local minimum than questing after an unreachable global minimum

Was particle roaming addressed: Fraction of swarm out of bounds

F12, $n_x = 1000$, 13 of 20 functions



F3, $n_x = 1000$, 6 of 20 functions



Observations:

- Larger seed set sizes perform better as problem dimensionality increases
- The subspace initialization method performs better than uniform random initialization and initialization by non-linear simplex
- CVT performs little better than the subspace-based method for lower dimension, but is computationally expensive, especially for large dimensions
- Even the best initialization strategies only reduced particle roaming by more than 75% on 6 of the 20 benchmark functions

Discrete-Valued Optimization Problems

Binary PSO

- What is the problem?
  - PSO originally developed for optimizing continuous-valued variables
  - Uses vector algebra on floating-point vectors
- How to adapt PSO for binary-valued variables?
  - Binary PSO (binPSO) of Kennedy and Eberhart
  - Velocity remains a floating-point vector, but meaning changes
  - Velocity is no longer a step size, but is used to determine a probability of selecting bit 0 or bit 1
  - Position is a bit vector, i.e. $x_{ij} \in \{0, 1\}$
  - How to interpret velocity as a probability?

$$p_{ij}(t) = \frac{1}{1 + e^{-v_{ij}(t)}}$$

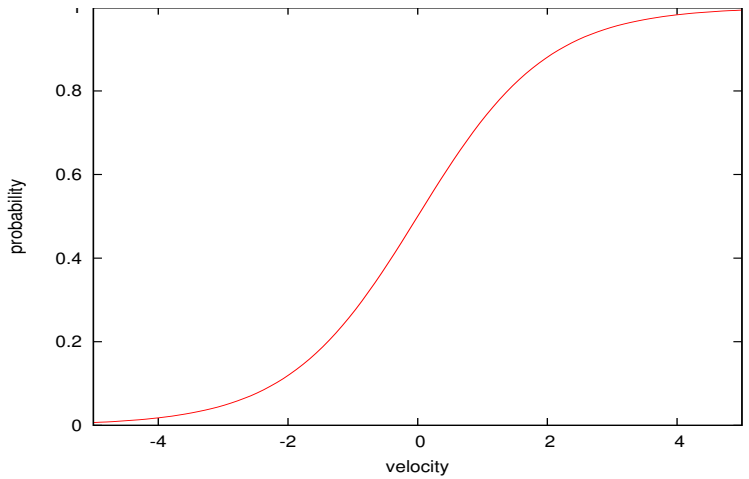
  - Then, position update changes to

$$x_{ij}(t+1) = \begin{cases} 1 & \text{if } U(0,1) < p_{ij}(t+1)) \\ 0 & \text{otherwise} \end{cases}$$

Issues:

- Interpretation of control parameters changes
  - w: small values facilitate longer exploration
  - V_{max}: smaller values promote exploration
- Initial velocities should be zero
- Velocities should never move to zero, but to $\pm\infty$
- Curse of dimensionality

Sigmoid Function:



Discrete-Valued Optimization Problems

Angle Modulated PSO[20]

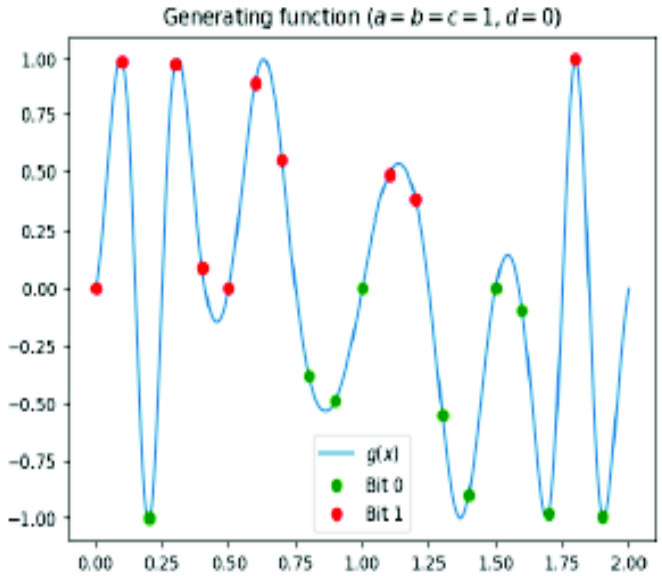
- Velocities and positions remain floating-point vectors
- Find a bitstring generating function to generate bitstring solution:

$$g(x) = \sin(2\pi(x-a) \times b \times \cos(2\pi(x-a) \times c)) + d$$

sampled at evenly spaced positions, x

The coefficients determine the shape of the generating function:

- a: horizontal shift of generating function
- b: maximum frequency of the sin function
- c: frequency of the cos function
- d: vertical shift of generating function



[20] G Pampara, N Franken, AP Engelbrecht, *Combining Particle Swarm Optimisation with Angle Modulation to Solve Binary Problems*, IEEE Congress on Evolutionary Computation, 2005

Use a standard PSO to find the best values for these coefficients
Generate a swarm of 4-dimensional particles;
repeat
    Apply any PSO for one iteration;
    for *each particle* **do**
        Substitute values for coefficients a, b, c and d into generating function;
        Produce n_x bit-values to form a bit-vector solution;
        Calculate the fitness of the bit-vector solution in the original bit-valued space;
    end
until *a convergence criterion is satisfied*;

Discrete-Valued Optimization Problems

Set-based PSO

Various PSO algorithms have been developed to solve problems where decision variables are discrete-valued:

- Binary PSO for when $x_{ij} \in \mathbb{R}$
- Angle modulated PSO which solves binary-valued optimization problems through a homomorphous mapping to a 4-dimensional continuous-valued space
- PSO for solving integer-valued optimization problems, wheren $x_{ij} \in \mathbb{Z}$

Of interest for this lecture, is set-based PSO algorithms, where

- particle positions are sets, and
- velocities are sets of operation pairs, specifying operations to be applied to particle position sets

Discrete-Valued Optimization Problems

Set-based PSO (cont)

- Discrete PSO by Correa *et al* to perform feature selection on bioinfomratics data
- SetPSO by Neethling and Engelbrecht to predict RNA secondary structures
- Set swarm optimization by Veenhuis for data clustering
- Set-based PSO by Chen *et al* for combinatorial optimization problems, specifically the traveling salesman and multi-dimensional knapsack problems
- Fuzzy PSO by Khan and Engelbrecht to solve the multi-objective problem of finding optimal topologies for distributed local area networks
- Fuzzy evolutionary PSO by Mohiuddin *et al* to solve the shortest path first weight setting problem
- Rough set-based PSO by Fen *et al* where particles represent rough sets

These early approaches have drawbacks:

- Many are domain specific, with domain specific operators
- Some do not truly reflect mathematical sets
- Some adjusts the elements in the sets, and not the sets themselves
- Some use fixed-size sets

These drawbacks limit their applicability to set-based optimization problems in general

Langeveld and Engelbrecht[21] proposed a generic set-based PSO:

- Does not use domain specific operators
- Elements within sets and sets themselves are adjusted
- Sets of varying sizes are allowed

[21] *Set-Based Particle Swarm Optimization applied to the Multidimensional Knapsack Problem*, Swarm Intelligence, 6(4):297–342, 2012

Discrete-Valued Optimization Problems

Set-based PSO (cont)

The set-based PSO (SBPSO) has since been applied to solve the following single-objective optimization problems:

- Multi-dimensional knapsack problems
- Feature selection
- Portfolio optimization
- Data clustering
- Rule induction

The following bi-level optimization problems:

- Polynomial approximation
- Training support vector machines

The following multi-objective optimization problems:

- Portfolio optimization
- Rule induction
- Multi-dimensional knapsack problems

Consider the following definitions:

- Let $U = \bigcup_{n=1}^{l} e_n$ be the universal set containing l elements
- e_n is a single element of the universe
- $\mathscr{P}(U)$ is the power set of U, i.e. the set of all subsets
- Particle position: $X_i(t) \subseteq \mathscr{P}(U)$
- $(\pm, e_n)$ is an operation pair, which represents either addition or removal of an element from $X_i(t)$
- Particle velocity: $V_i(t) = \{v_{i,1}, ..., v_{i,k}\} = \{(\pm, e_{n_{i,1}}), ..., (\pm, e_{n_{i,k}})\}$
- Personal best position: $Y_i(t)$
- Neighbourhood best position: $\widehat{Y}_i(t)$

Discrete-Valued Optimization Problems

Set-based PSO (cont)

Particle position update equation:

$$X_i(t+1) = X_i(t) \boxplus V_i(t+1)$$

Velocity update equation:

$$V_i(t+1) = c_1 r_1 \otimes (Y_i(t) \ominus X_i(t)) \oplus c_2 r_2 \otimes (\widehat{Y}_i(t) \ominus X_i(t)) \\ \oplus (c_3 r_3 \odot_k^+ A_i(t)) \oplus (c_4 r_4 \odot^- S_i(t))$$

where

- $r_i \sim U(0,1)$ for $i = 1, ..., 4$
- c_1 determines attraction to the personal best position
- c_2 determines attraction to the neighbourhood best position
- c_3 and c_4 are positive acceleration constants for the addition and removal of elements, respectively

Addition of velocities:

- the mapping $\oplus : \mathscr{P}(\{+,-\} \times U)^2 \rightarrow \mathscr{P}(\{+,-\} \times U)$
- defines the addition of two velocities to produce a new velocity
- simply the union of the two sets of operation pairs:

$$V_1 \oplus V_2 = V_1 \cup V_2$$

- the union of operation pairs

Difference between particle positions:

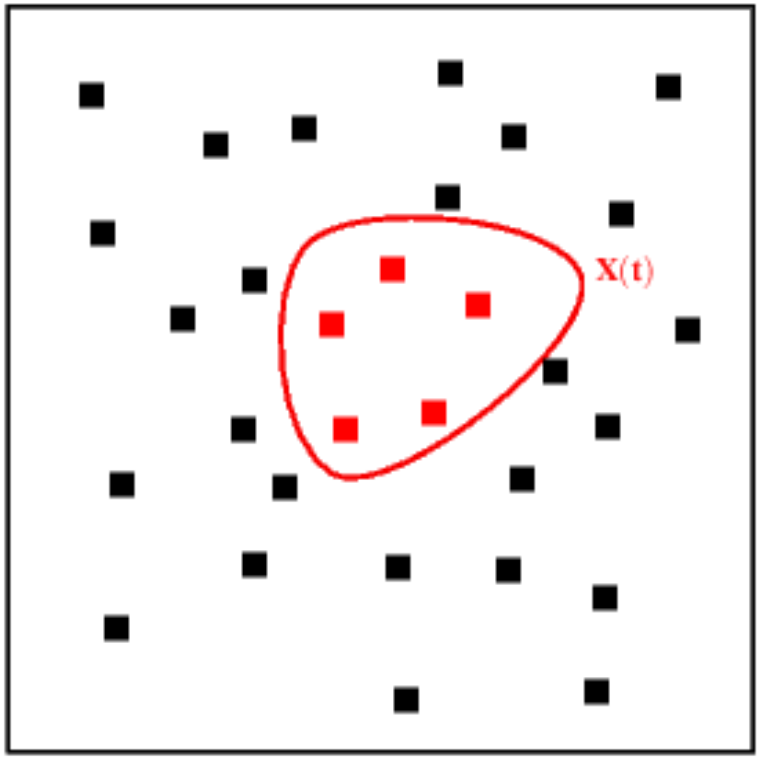
- the mapping $\ominus : \mathscr{P}(U)^2 \rightarrow \mathscr{P}(\{+,-\} \times U)$
- takes two positions as input and yields a velocity component
- resulting velocity represents the addition and removal operations required to convert X_2 into X_1:

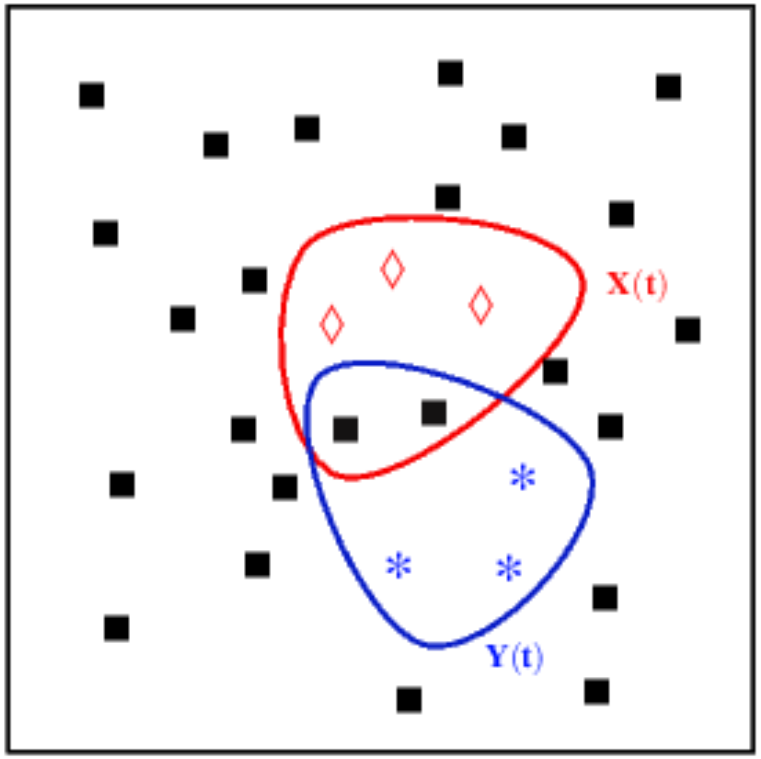
$$X_1 \ominus X_2 = (\{+\} \times (X_1 \setminus X_2)) \cup (\{-\} \times (X_2 \setminus X_1))$$

- simply put, $X_1 \ominus X_2$ is the union of
  - (i) the addition of all elements in X_1 not in X_2, and
  - (ii) the removal of all elements in X_2 not in X_1

Discrete-Valued Optimization Problems

Set-based PSO (cont)



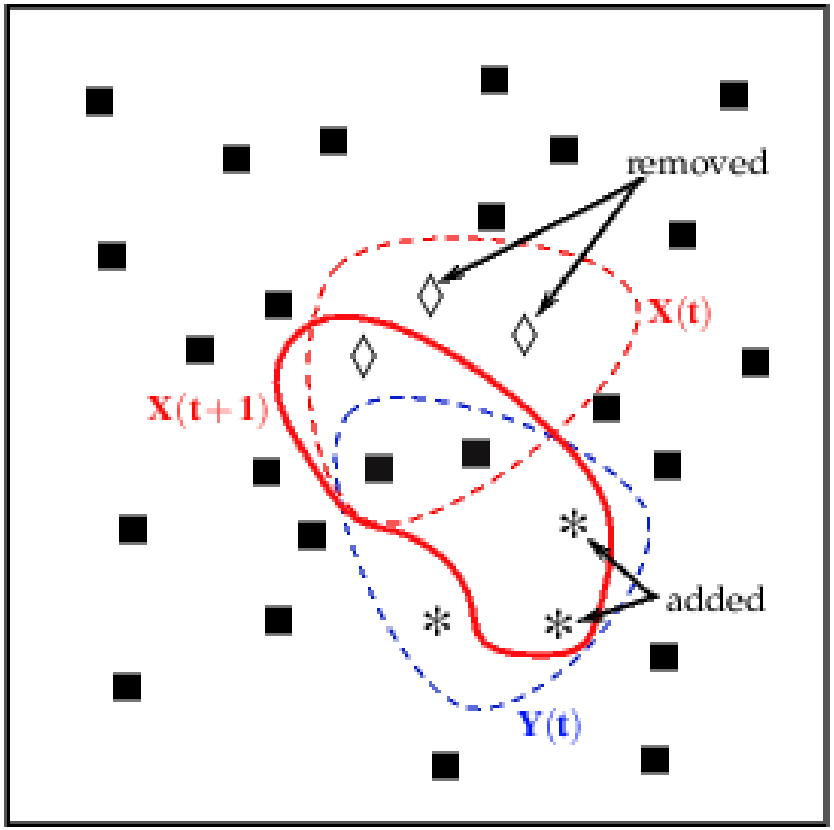


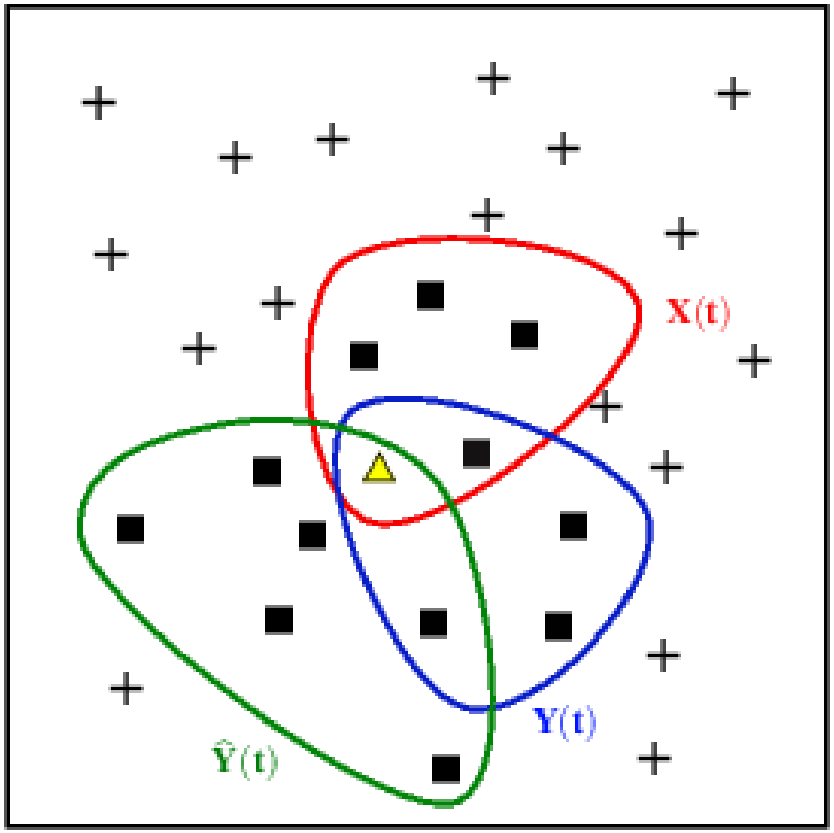
Moving $X(t)$ closer to $Y(t)$:

- remove elements from $X(t)$ that are not in $Y(t)$, i.e. $\Diamond$ elements
- add to $X(t)$ elements that are in $Y(t)$, i.e. $*$ elements

Discrete-Valued Optimization Problems

Set-based PSO (cont)





Multiplication of velocity by a scalar:

- the mapping $\otimes : [0, 1] \times \mathscr{P}(\{+, -\} \times U) \rightarrow \mathscr{P}(\{+, -\} \times U)$
- takes a scalar and a velocity and outputs a velocity
- operation $\eta \otimes V$, with $\eta \in [0, 1]$, means selection of a random subset of $\lfloor \eta \times |V| \rfloor$ elements from velocity V, to produce a new velocity
- note that $0 \otimes V = \emptyset$ and $1 \otimes V = V$

Addition of velocity and position:

- the mapping $\boxplus : \mathscr{P}(U) \times \mathscr{P}(\{+,-\} \times U) \rightarrow \mathscr{P}(U)$
- takes a position and a velocity, and yields a position
- $X \boxplus V$ is the application of the velocity V, i.e. a set of operation pairs, on the position X

$$X \boxplus V = V(X)$$

Removal of special elements:

- removal is denoted by $\beta \odot^{-} S$
- S is shorthand for $X(t) \cap Y(t) \cap \hat{Y}(t)$
- elements in $X(t) \cap Y(t) \cap \hat{Y}(t)$ are removed from $X(t)$
- the operator $\odot^{-} : [0, |S|] \times \mathscr{P}(U) \rightarrow \mathscr{P}(\{+, -\} \times U)$
  - takes a scalar and set of elements as input and yields a velocity
  - a random number of elements determined by β are selected from S for removal from $X(t)$
- the result is a set of deletions, i.e. a velocity

$$\beta \odot^{-} S = \{-\} \times \left(\frac{N_{\beta,S}}{|S|} \otimes S \right)$$

where $N_{\beta,S} = \min(|S|, \lfloor \beta \rfloor + \mathcal{I}_{\{r<\beta-\lfloor \beta \rfloor\}})$; $\beta \in \mathbb{R}^{+}$ and $r \sim U(0,1)$

- the indicator function $\mathcal{I}_{\{bool\}}$ has a value of 1 if *bool = true* and 0 if *bool = false*

Addition of special elements:

- addition is denoted by $\beta \odot_k^+ A$
- A is shorthand for $U \setminus (X_i(t) \cup Y_i(t) \cup \widehat{Y}_i(t))$
- elements *not* in $X(t) \cup Y(t) \cup \widehat{Y}(t)$ are added to $X(t)$
- the operator $\odot^+ : [0, |A|] \times \mathscr{P}(U) \rightarrow \mathscr{P}(\{+, -\} \times U)$
  - takes a scalar and a set of elements as input and yields a set of additions, i.e. a velocity

$$\beta \odot_k^+ A = \{+\} \times k\text{-Tournament Selection}(A, N_{\beta,A})$$

k-Tournament Selection$(A, N_{\beta,A})$

Set $V_{temp} = \emptyset$;
for $n = 1, \ldots, N$ **do**
  for $j = 1, \ldots, k$ **do**
    Randomly select e_j from A;
    Set $score_j = f(X(t) \cup \{e_j\})$;
  end
  Select $m \in \{1, \ldots, k\}$ such that $score_m = \max_j \{score_j\}$;
  Set $V_{temp} = V_{temp} \oplus (\{+\} \times e_m)$;
end
Return V_{temp};

The number of additions $N_{\beta,A}$

- determined by β, in a similar manner as the removal
- elements are selected from A using tournament selection with a tournament size of k particles

Niching capability of PSO:

- Can the *gbest* PSO find more than one solution?
  - Formal proofs showed that all particles converge to a weighted average of their personal best and global best positions
  - Therefore, only one solution can be found
  - What if we re-run the algorithm? No guarantee to find different solutions
- What about *lbest* PSO?
  - Neighborhoods may converge to different solutions
  - However, due to overlapping neighborhoods, particles are still attracted to one solution

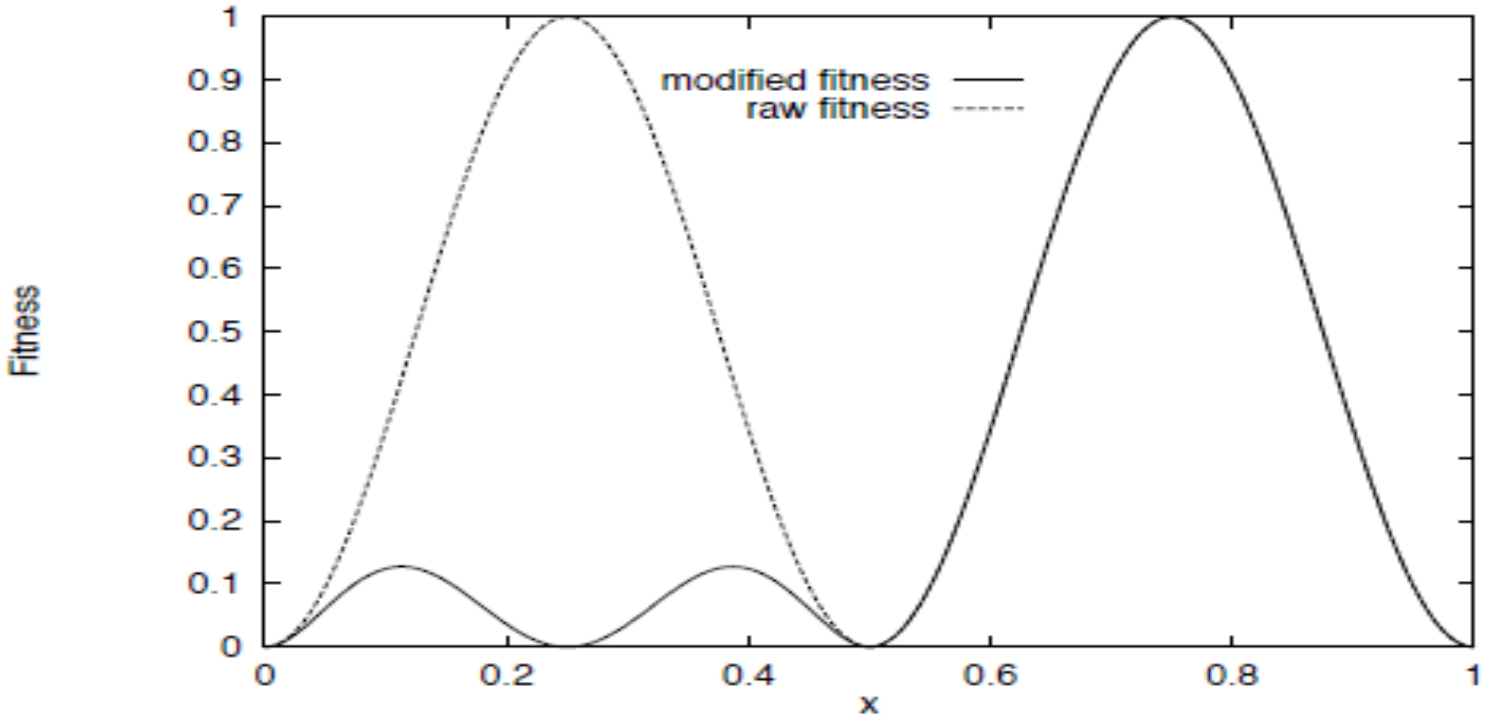
Multiple Solutions to Multimodal Problems

Objective Function Stretching

Sequential niching, stretching the function to remove found minima

Create and initialize a n_x-dimensional swarm, S, and $\mathcal{X} = \emptyset$;
repeat
    Perform a single PSO iteration;
    if $f(S.\hat{\mathbf{y}}) \leq \epsilon$ **then**
        Isolate $S.\hat{\mathbf{y}}$;
        Perform a local search around $S.\hat{\mathbf{y}}$;
        if *a minimizer* $\mathbf{x}^*_{\mathcal{N}}$ *is found by the local search* **then**
            $\mathcal{X} \leftarrow \mathcal{X} \cup \{\mathbf{x}^*_{\mathcal{N}}\}$;
            Let $f(\mathbf{x}) \leftarrow H(\mathbf{x})$;
        end
    end
    Reinitialize the swarm S;
until *stopping condition is true*;
Return $\mathcal{X}$ as the set of multiple solutions;

Assume maximization



Effect of Sequential Niching for One Dimension

Parallel niching PSO

Create and initialize a n_x-dimensional *main* swarm, S;
repeat
    Train main swarm, S, for one iteration using *cognition-only* model;
    Update the fitness of each main swarm particle, $S.\mathbf{x}_i$;
    for *each sub-swarm* S_k **do**
        Train sub-swarm particles, $S_k.\mathbf{x}_i$, using a full model PSO;
        Update each particle's fitness;
        Update the swarm radius $S_k.R$;
    endFor
    If possible, merge sub-swarms;
    Allow sub-swarms to absorb any particles from the main swarm that moved into the sub-swarm;
    If possible, create new sub-swarms;
until *stopping condition is true*;

Multiple Solutions to Multimodal Problems

NichePSO

Although the NichePSO was shown to accurately and efficiently locate multiple optima, it has the following problems:

- Finds and maintains niches in parallel
- Not much scalable, as it requires a swarm size that grows exponentially with the number of optima to be found
- Main reason for this is that particles waste time re-exploring areas of the search space already explored

To address these problems, the NichePSO is combined with a sequential niching method to prevent re-exploration of the search space

Multiple Solutions to Multimodal Problems

Derating NichePSO

- Derating NichePSO prevents re-exploration of the search space by hybridizing the NichePSO with the sequential niching technique (SNT)
- SNT is a sequential niching method originally developed for genetic algorithms
- Makes use of knowledge gained about the search space to prevent re-exploration
- Knowledge gained about the search space is reflected in two fitness functions:
  - The original function to be optimized, F, referred to as the *raw fitness function*
  - The *modified fitness function*, M, which is an adapted version of F with all found optima removed

- Whenever an optimum is found, the modified fitness functions is updated:

$$M_{t+1} = M_t(\mathbf{x}) * G(\mathbf{x}, \hat{\mathbf{y}}_t), \quad M_0(\mathbf{x}) = F(\mathbf{x})$$

- G is a single-peak derating function, and $\hat{\mathbf{y}}$ is the found optimum, e.g.

$$G(\mathbf{x}, \hat{\mathbf{y}}) = \begin{cases} e^{\log m(r - d_{\mathbf{x}\hat{\mathbf{y}}})/r} & \text{if } d_{\mathbf{x}\hat{\mathbf{y}}} < r \\ 1 & \text{otherwise} \end{cases}$$

where $d_{\mathbf{x}\hat{\mathbf{y}}}$ is the Euclidean distance between a candidate solution $\mathbf{x}$ and the found optimum $\hat{\mathbf{y}}$, and the niche radius is

$$r = \frac{\sqrt{n}}{2\sqrt[q]{n}}$$

where q is the number of optima, and n is the dimension of the search space

- The Derating NichePSO consists of two phases:
  - Phase 1: Applies NichePSO to the modified fitness function to form sub-swarms around potential solutions
  - Phase 2: Sub-swarms exploit solutions wrt the raw fitness function, which will allow sub-swarms to converge to true optima

Let $t = 0$ and $M_t = F$;
$\mathbf{X} = \{\}$;
while *stopping condition not met* **do**
    Initialize the NichePSO;
    Perform Phase 1;
    Perform Phase 2;
    Set $\mathbf{X} = \mathbf{X} \cup \{\text{NichePSO sub-swarms}\}$;
    Merge sub-swarms in $\mathbf{X}$;
    Set $M_{t+1} = F$;
    for *each sub-swarm in* $\mathbf{X}$ **do**
        Update M_{t+1} using the best particle in the sub-swarm as $\hat{\mathbf{y}}$;
    end
    $t = t + 1$;
end

Phase 1:

while *main swarm still has particles or maximum number of iterations has been exceeded* **do**
    Apply one iteration of the cognition-only model to main swarm particles wrt M;
    Update the fitness of each main swarm particle, using M;
    Create new sub-swarms if possible;
end

Multiple Solutions to Multimodal Problems

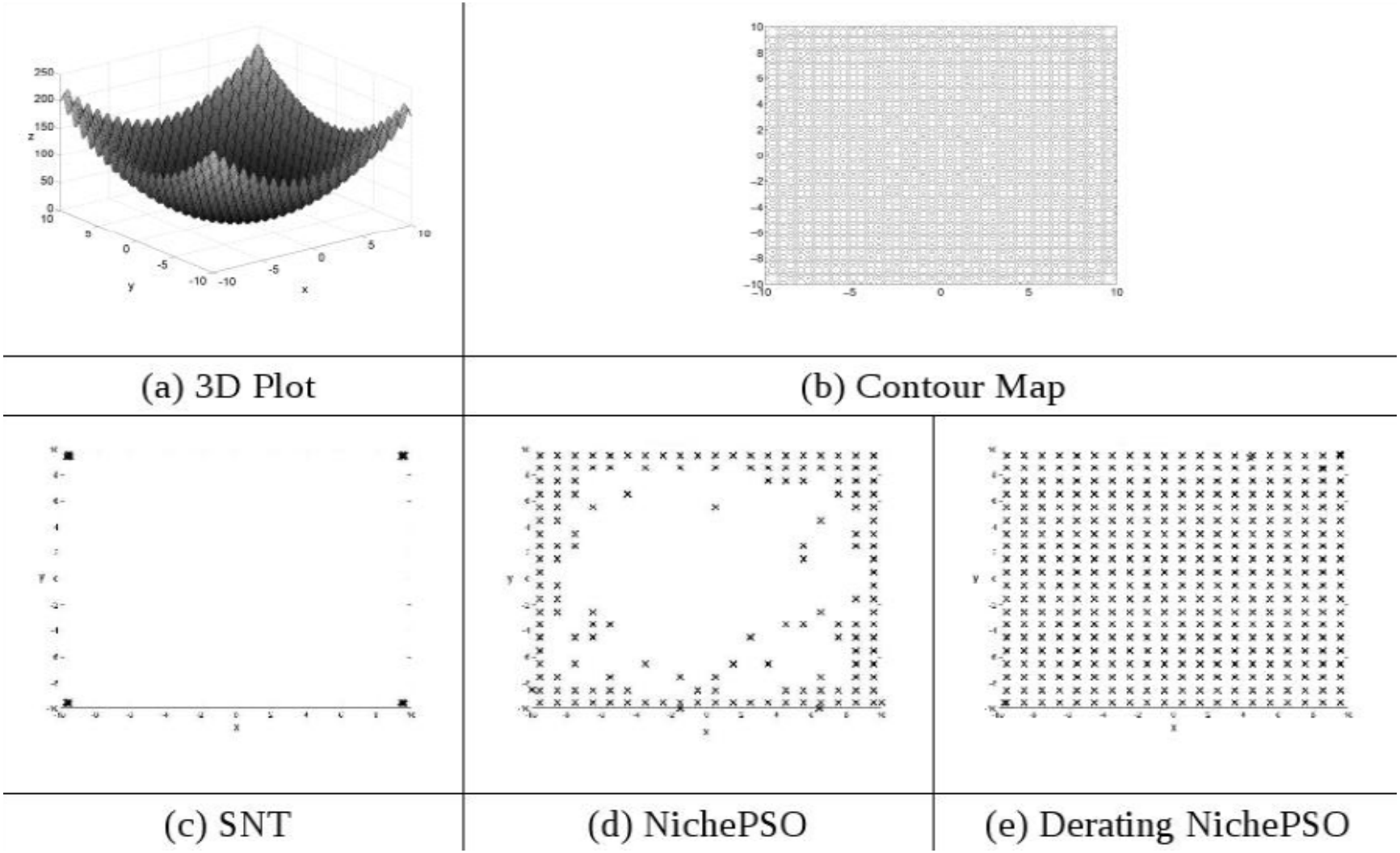
Derating NichePSO (cont)

repeat
  for *each sub-swarm* **do**
    Apply one iteration of the gbest PSO to the sub-swarm particles wrt F;
    Update each particle's fitness using F;
    Update sub-swarm radius;
  end
  Absorb main swarm particles if necessary;
  Create new sub-swarms if possible;
  Merge sub-swarms if necessary;
until *all sub-swarms have converged or maximum number of iterations has been exceeded*;

Multiple Solutions to Multimodal Problems

Derating NichePSO (cont)

Rastrigin function



Multiple Solutions to Multimodal Problems

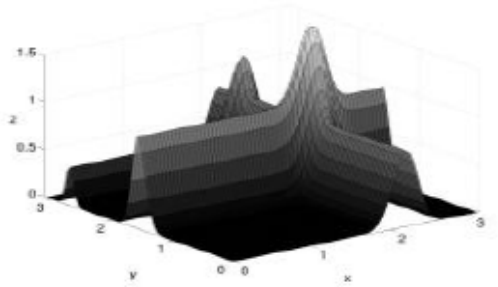
Derating NichePSO (cont)

Algorithm	Average Solutions	Std Dev	Average Accuracy	Std Dev
SNT	10.000	0.000	20.345	17.965
NichePSO	45.100	4.414	0.083	0.235
Derating NichePSO	132.00	6.140	0.183	0.007

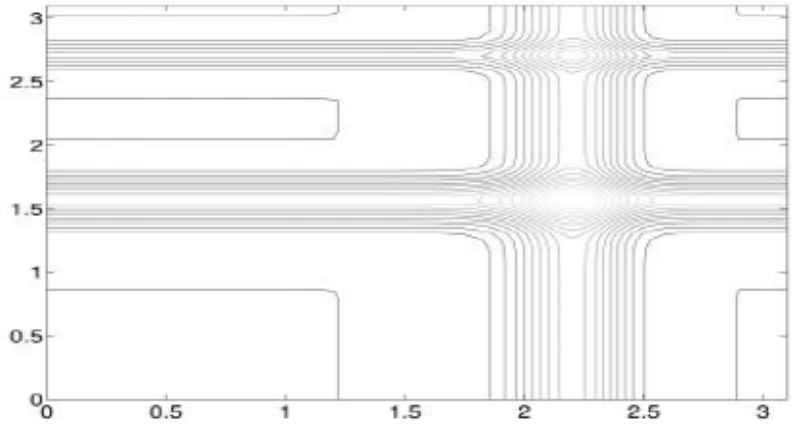
Multiple Solutions to Multimodal Problems

Derating NichePSO (cont)

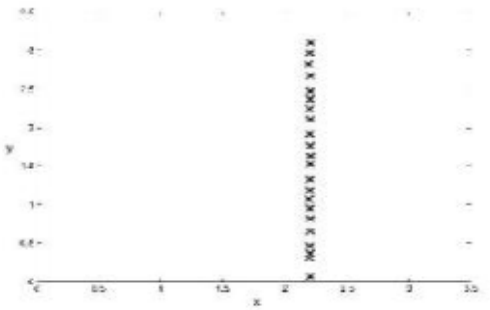
Michalewicz function



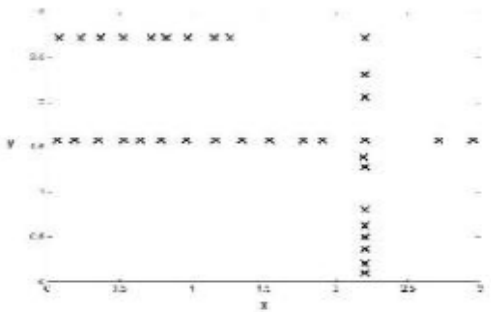
(a) 3-D Plot



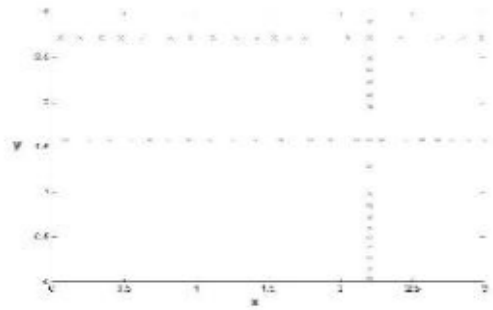
(b) Contour Plot



(c) SNT



(d) NichePSO



(e) Derating NichePSO

Multiple Solutions to Multimodal Problems

Derating NichePSO (cont)

Algorithm	Average Solutions	Std Dev	Average Accuracy	Std Dev
SNT	12.570	1.947	0.015	0.005
NichePSO	16.034	2.785	0.012	0.013
Derating NichePSO	72.967	4.874	0.005	0.004

Scalibility: A relationship between

- the swarm size, $|S|$,
- the number of times, R, that phase 1 is executed, and
- the number of solutions that can be found

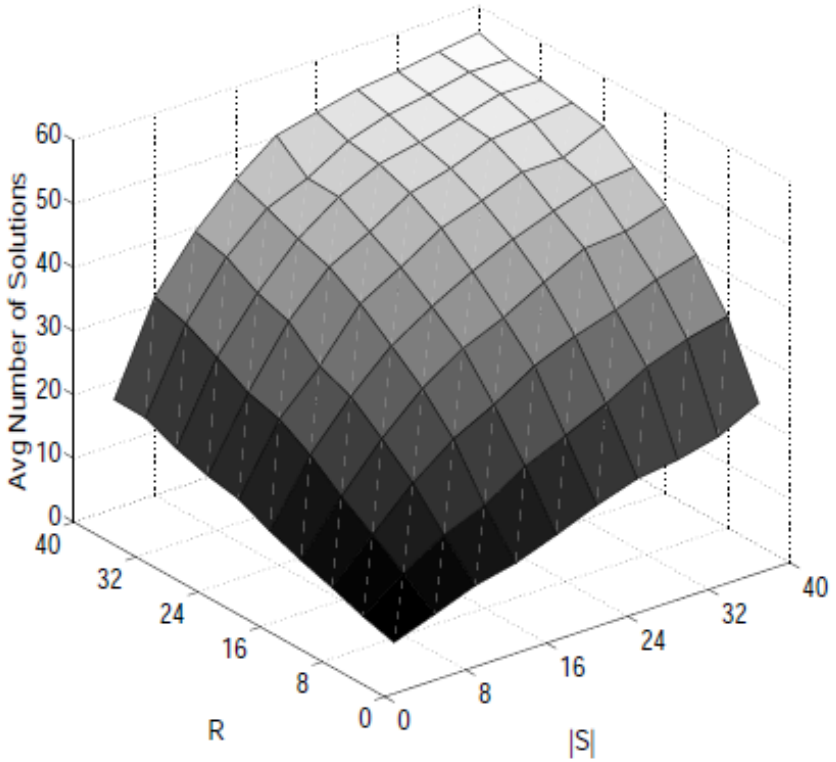
$$\overline{Solutions} \approx q \times \left(\frac{e^{\frac{\lambda}{q}(R \times |S|)} - e^{-\frac{\lambda}{q}(R \times |S|)}}{e^{\frac{\lambda}{q}(R \times |S|)} + e^{-\frac{\lambda}{q}(R \times |S|)}} \right)$$

where q is the maximum number of solutions, and λ is a measure of how convoluted the search space is

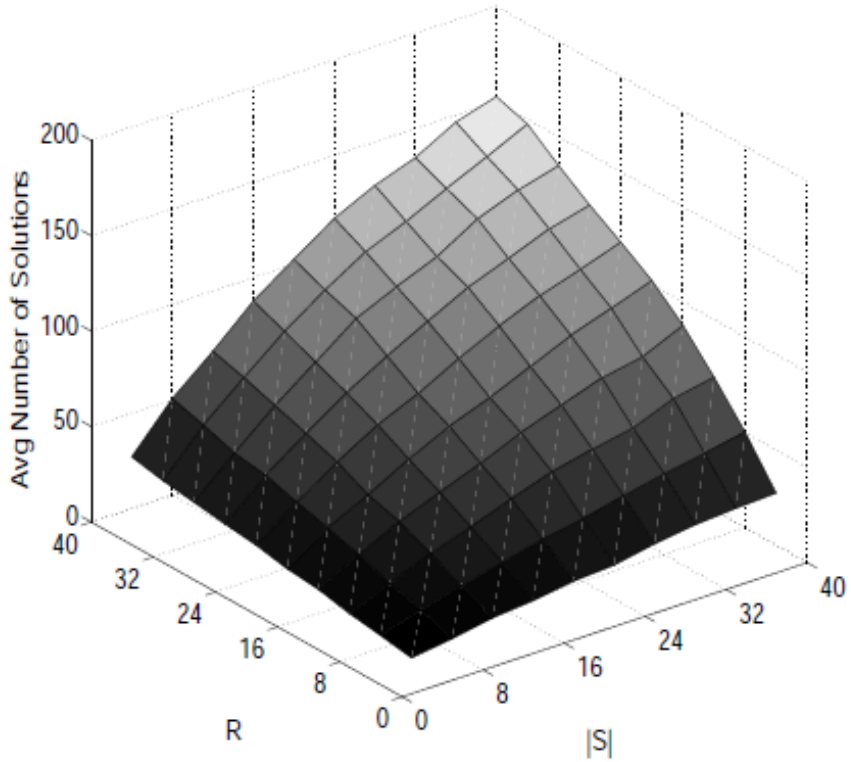
Multiple Solutions to Multimodal Problems

Derating NichePSO (cont)

Rastrigin



Michalewicz



The speciation PSO has the following steps:

1. Generate an initial swarm with randomly generated particles
2. Evaluate all particles in the swarm
3. Sort all particles in descending order of their fitness value
4. Assign each species seed identified as the nbest to all particles identified in the same species
5. Adjust each particle's velocity and position
6. If termination criterion is not met, goto step 2

Algorithm for determining species seeds

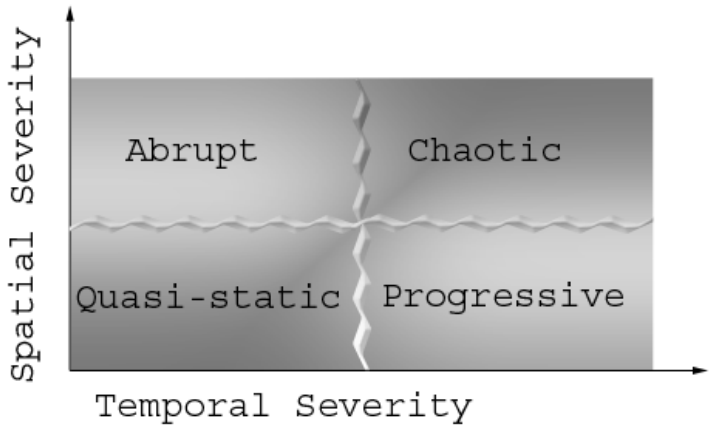
1. Let L_{sorted} = all particles sorted in decreasing order of fitness.
2. Let $S_{seed} = \{\}$
3. While L_{sorted} contains *unprocessed* particles do
   - **a.** Get the first *unprocessed* particle $\mathbf{x}_L$ in L_{sorted}
   - **b.** Let $found = FALSE$
   - **c.** For each particle $\mathbf{x}_S$ in S_{seed} do
     - **c.1** If $d(\mathbf{x}_S, \mathbf{x}_L) \leq r_s$ then
       - **c.1.2** $found = TRUE$
       - **c.1.3** Goto step *d*.
   - **d.** If (not $found$) then let $S_{seed} = S_{seed} \cup \{\mathbf{x}_L\}$
   - **e.** Mark $\mathbf{x}_L$ as *processed*.

- Objective: To find and track solutions in dynamically changing search spaces

$$\mathbf{x}^*(t) = \min_{\mathbf{x}} f(\mathbf{x}, \varpi(t))$$

where $\mathbf{x}^*(t)$ is the optimum found at time step t, and $\varpi(t)$ is a vector of time-dependent objective function control parameters

- Environment types:
  - Location of optima may change
  - Value of optima may change
  - Optima may disappear and new ones appear
  - Change frequencey
  - Change severity



Environment Classes

Dynamic Environments

Consequences for PSO

- PSO can not be applied to dynamic environments without any changes to maintain swarm diversity
- Recall that particles converge to a weighted average of their personal best and global best positions
- At the point of convergence, $\mathbf{v}_i = 0$, and the contributions of the cognitive and social components are approximately zero
- New velocities are zero, therefore no change in position
- When the environment changes, personal best positions becomes stale, and will cause particles to be attrackted to old best positions
- Small inertia weight values limit exploration
- Velocity clamping limits exploration

Dynamic Environments

Consequences for PSO (cont)

- Environment change detection:
  - Optimization algorithm needs to react when a change is detected in order to increase diversity
  - Use sentry particles
  - Gbest versus pbest versus arbitrary positions as sentries
- How to respond to environment changes?
  - Change the inertia update
    - If decreasing inertia is used, reset w to larger value

Dynamic Environments

Consequences for PSO (cont)

- Reinitialize particle positions:
  - Reinitialize the entire swarm
  - Reinitialize parts of the swarm
  - Total reinitialization versus keeping previous personal best positions
- Limit memory
  - Reinitialize the personal best position to the particle's current position – only effective if swarm has not yet converged
  - Reset personal best positions only if significant change in fitness is observed
  - Re-evaluate best positions
  - Recalculate global best after resetting personal best positions
- Do a local search around the previous optimum

- Some particles attract one another, and others repell one another
- Velocity changes to

$$v_{ij}(t+1) = wv_{ij}(t)+c_1 r_1(t)[y_{ij}(t)-x_{ij}(t)]+c_2 r_2(t)[\hat{y}_j(t)-x_{ij}(t)]+a_{ij}(t)$$

where $\mathbf{a}_i$ is the particle acceleration, determining the magnitude of inter-particle repulsion

$$\mathbf{a}_i(t) = \sum_{l=1, i\neq l}^{n_s} \mathbf{a}_{il}(t)$$

- The repulsion force between particles i and l is

$$\mathbf{a}_{il}(t) = \begin{cases} \left(\frac{Q_i Q_l}{d_{il}^3}\right)(\mathbf{x}_i(t) - \mathbf{x}_l(t)) & \text{if } R_c \leq d_{il} \leq R_p \\ 0 & \text{otherwise} \end{cases}$$

Dynamic Environments

Charged PSO (cont)

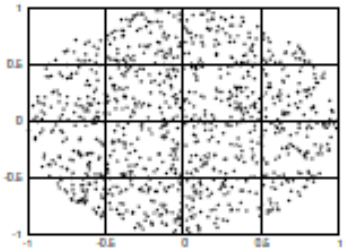
- $d_{il} = ||\mathbf{x}_i(t) - \mathbf{x}_l(t)||$
  Q_i is the particle's charged magnitude
  R_c is the core radius
  R_p is the perception limit of each particle
- If $Q_i = 0$, particles are neutral and there is no repelling
- If $Q_i \neq 0$, particles are charged, and repel from each other
- Inter-particle repulsion occurs only when the separation between two particles is within the range $[R_c, R_p]$
- The smaller the separation, the larger the repulsion between the corresponding particles
- Acceleration is fixed at the core radius to prevent too severe repelling

- Based on quantum model of an atom, where orbiting electrons are replaced by a quantum cloud which is a probability distribution governing the position of the electron
- Developed as a simplified and less expensive version of the charged PSO
- Swarm contains
  - neutral particles following standard PSO updates
  - charged, or quantum particles, randomly placed within a multi-dimensional sphere

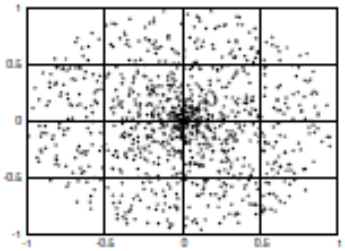
$$\mathbf{x}_i(t+1) = \begin{cases} \mathbf{x}_(t) + \mathbf{v}_i(t+1) & \text{if } Q_i = 0 \\ \mathbf{B}_{\hat{\mathbf{y}}}(r_{cloud}) & \text{if } Q_i \neq 0 \end{cases}$$

Dynamic Environments

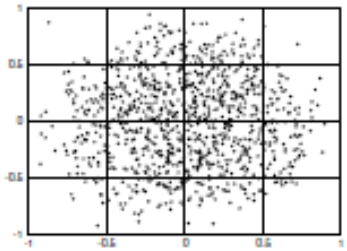
Quantum-inspired PSO: Different Distributions[22]



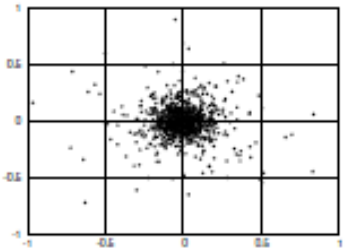
(a) Uniform



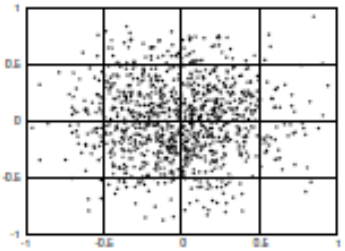
(b) Non-uniform



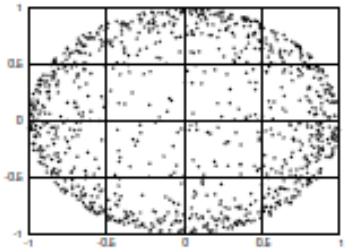
(c) Gaussian



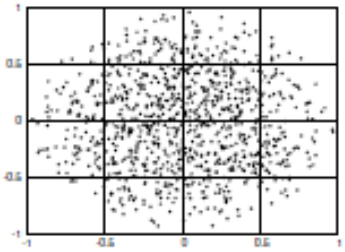
(d) Cauchy



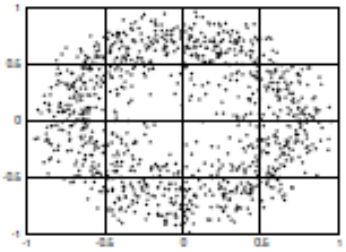
(e) Exponential



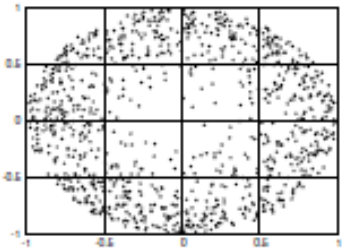
(f) Beta



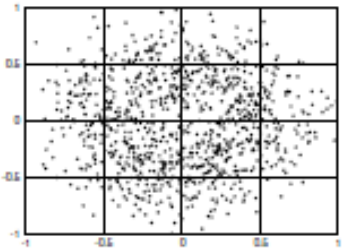
(g) Triangular-0



(h) Triangular-0.5



(i) Triangular-1



(j) Weibull

[22] KR Harrison, BM Ombuki-Berman, AP Engelbrecht, *The Effect of Probability Distributions on the Performance of Quantum Particle Swarm Optimization*, IEEE Swarm Intelligence Symposium 2015

Constrained Optimization Problems

Coevolutionary PSO

- Define the Lagrangian for the constrained problem
- The new optimization problem:

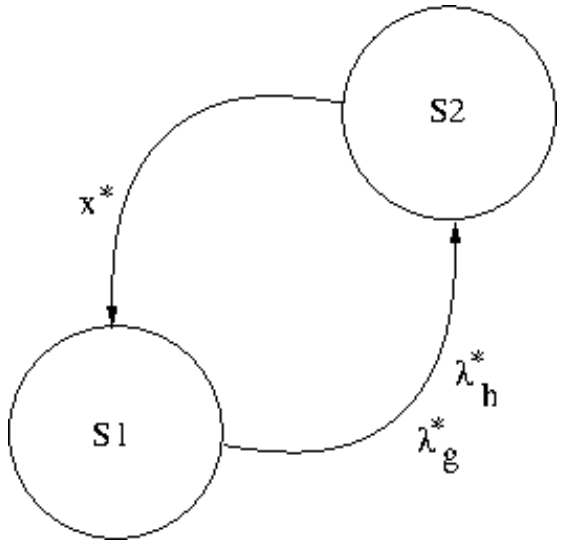
$$\min_{\mathbf{x}} \max_{\lambda_g, \lambda_h} L(\mathbf{x}, \lambda_g, \lambda_h)$$

- A coevolutionary PSO approach to solve the above min-max problem uses two swarms
- Swarm S_1 uses fitness function

$$f(\mathbf{x}) = \max_{\lambda_g, \lambda_h \in S_2} L(\mathbf{x}, \lambda_g, \lambda_h)$$

- Swarm S_2 uses fitness function

$$f(\lambda_g, \lambda_h) = \min_{\mathbf{x} \in S_1} L(\mathbf{x}, \lambda_g, \lambda_h)$$



Create and initialize two swarms, S_1 and S_2, where S_1 is n_x-dimensional and S_2 is $n_g + n_h$ dimensional;
repeat
  Run a PSO algorithm on swarm S_1 for $S_1.n_t$ iterations;
  Re-evaluate $S_2.\mathbf{y}_i(t), \forall i = 1, \ldots, S_2.n_s$;
  Run a PSO algorithm on swarm S_2 for $S_2.n_t$ iterations;
  Re-evaluate $S_1.\mathbf{y}_i(t), \forall i = 1, \ldots, S_1.n_s$;
until *stopping condition is true*;

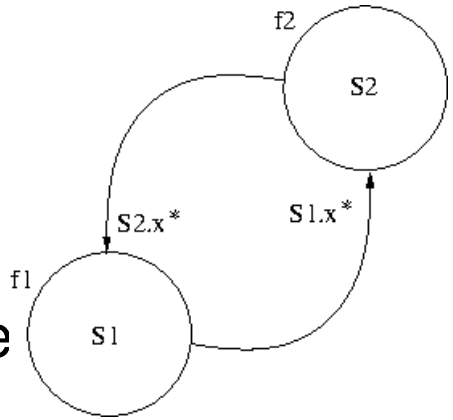
Transform the constrained problem to

- an unconstrained problem using a penalty method
  - solve the unconstrained problem using any PSO for boundary constrained problems
  - note the issues with penalty approaches
- a multi-objective optimization problem
  - reformulate constraints as additional sub-objective(s)
  - solve using a multi-objective PSO algorithm

- Convert the multi-objective problem into a single-objective problem
- Solve using a PSO for boundary constrained single-objective problems
- Note the issues with weighted aggregation methods

Multi-Objective Optimization

Vector-Evaluated PSO



A multi-swarm approach:

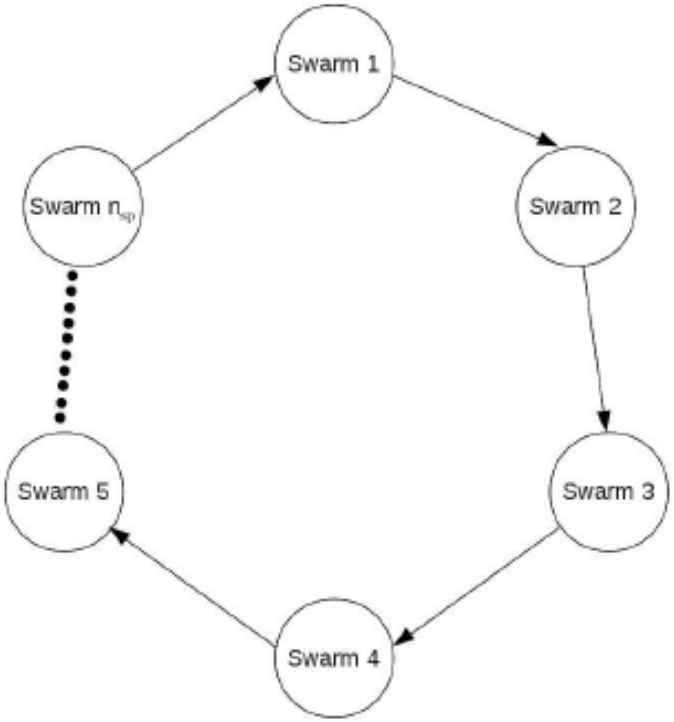
- Assume K sub-objectives
- K sub-swarms are used, where each optimizes one objectives
- Need a knowledge transfer strategy (KTS) to transfer information about best positions between sub-swarms
- Exchanged information are via selection of global guides, replacing the global best positions in the velocity updates
- Standard KTS: the ring KTS
  - Sub-swarms are arranged in a ring topology
  - Global guide of swarm S_k is swarm $S_{(k+1) \mod K}$

Multi-Objective Optimization

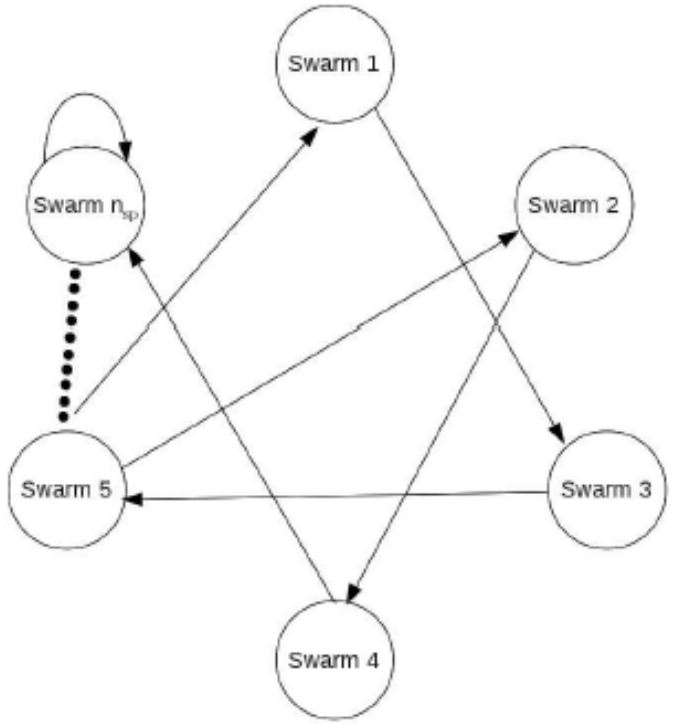
Vector-Evaluated PSO: KTS[23]

Example KTSs:

- Ring KTS
- Random KTS
- Parent-centric crossover archive-based KTS



(a) Ring Topology



(b) Random Topology

- Assume two objectives and ring KTS:

$$\begin{aligned} S_1.v_{ij}(t+1) &= wS_1.v_{ij}(t) + c_1 r_{1j}(t)(S_1.y_{ij}(t) - S_1.x_{ij}(t)) \\ &+ c_2 r_{2j}(t)(S_2.\hat{y}_i(t) - S_1.x_{ij}(t)) \\ S_2.v_{ij}(t+1) &= wS_2.v_{ij}(t) + c_1 r_{1j}(t)(S_2.y_{ij}(t) - S_2.x_{ij}(t)) \\ &+ c_2 r_{ij}(t)(S_1.\hat{y}_j(t) - S.x_{2j}(t)) \end{aligned}$$

  where sub-swarm S_1 evaluates individuals on the basis of objective $f_1(\mathbf{x})$, and sub-swarm S_2 uses objective $f_2(\mathbf{x})$
- Local guide selection:
  - Local guide replaces the personal best
  - Update personal best position only if the new particle position dominates the previous personal best position
- Alternative KTS: random

Multi-Objective Optimization

Using Archives

- Objective of archive is to keep track of all non-dominated solutions
- Non-dominated solutions added to archive after each iteration
- Fixed-sized archives versus unlimited sizes
- Local versus global guides

Let $t = 0$;
Initialize the swarm, $S(t)$, and archive, $A(t)$;
repeat
  Evaluate $(S(t))$;
  $A(t+1) \leftarrow$ Update$(S(t), A(t))$;
  $S(t+1) \leftarrow$ Generate$(S(t), A(t))$;
  $t = t + 1$;
until *stopping condition is true*;

VEPSO is computationally efficient, and simple, and has shown good promise

However, VEPSO was shown to perform bad for some problems

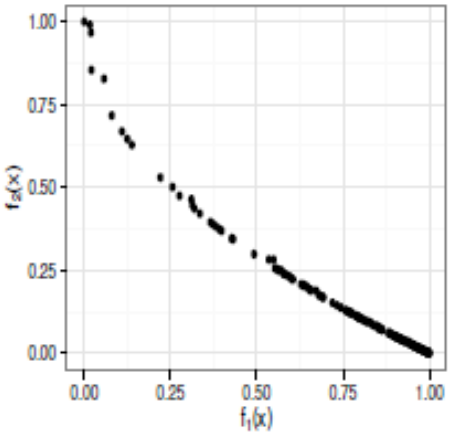
Recent analysis of VEPSO showed[24]

- a lack of exploitation of objective space,
- with lots of "exploration", mainly random movement in objective space

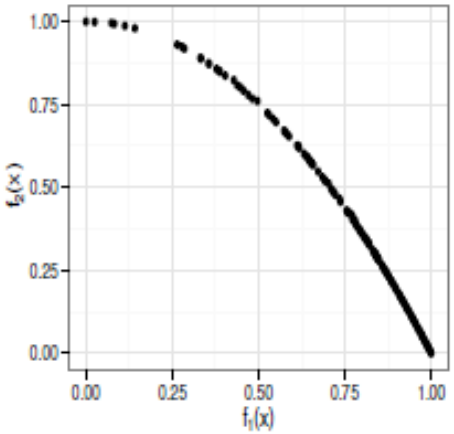
[24] C Scheepers, AP Engelbrecht, *Vector Evaluated Particle Swarm Optimization Part I: Explorative Analysis*, IEEE Congress on Evolutionary Computation, 2016; C Scheepers, AP Engelbrecht, *Vector Evaluated Particle Swarm Optimization Exploration behavior Part II: Quantitative Analysis*, IEEE Congress on Evolutionary Computation, 2016

Multi-Objective Optimization

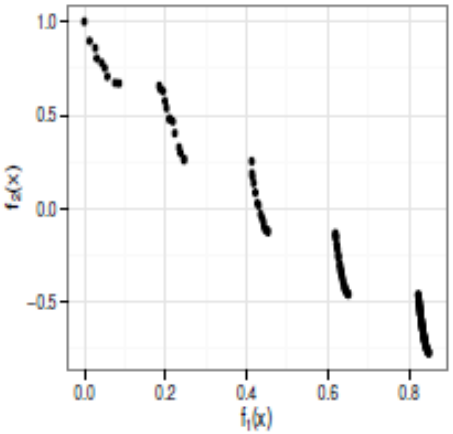
Vector-Evaluated PSO: Example Pareto Fronts: ZDT Problems



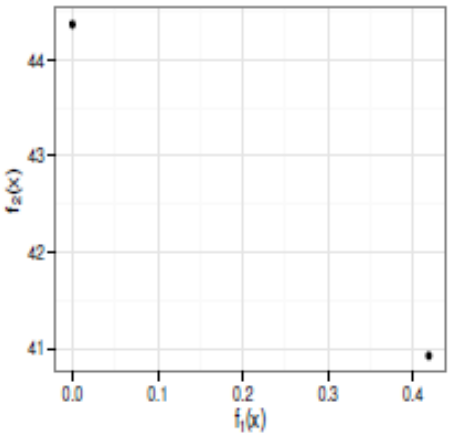
(a) ZDT1



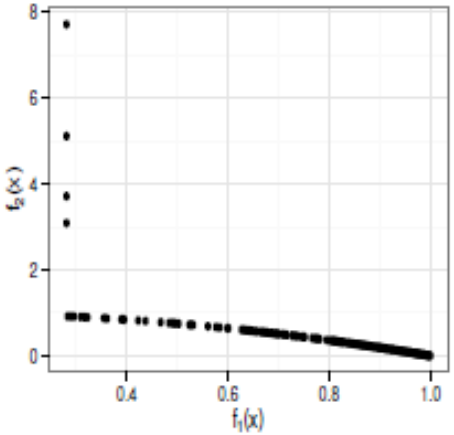
(b) ZDT2



(c) ZDT3



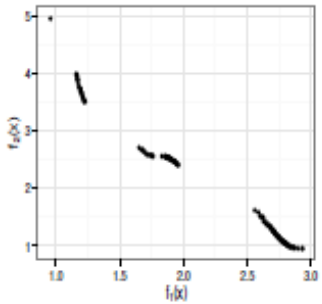
(d) ZDT4



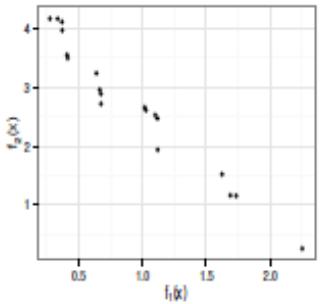
(e) ZDT6

Multi-Objective Optimization

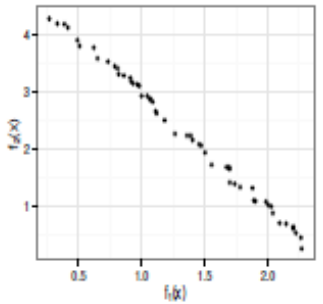
Vector-Evaluated PSO: Example Pareto Fronts: WFG Problems



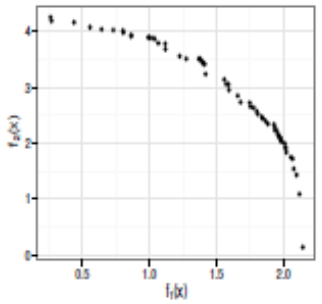
(a) WFG1



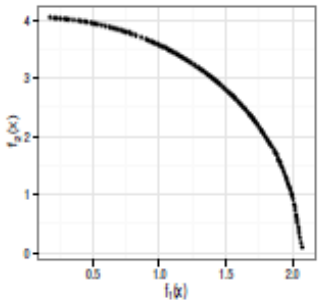
(b) WFG2



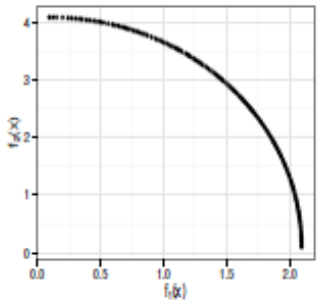
(c) WFG3



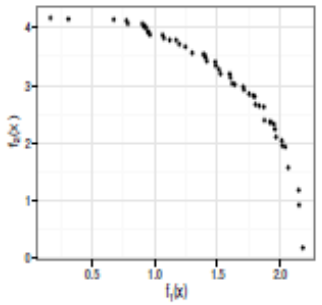
(d) WFG4



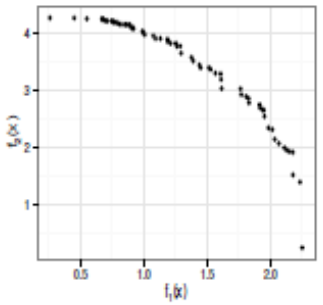
(e) WFG5



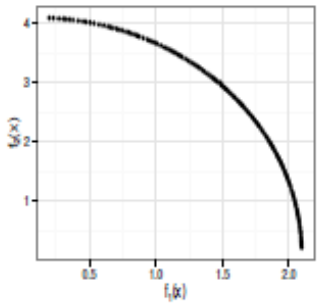
(f) WFG6



(g) WFG7



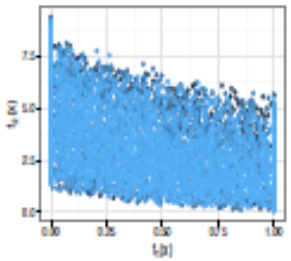
(h) WFG8



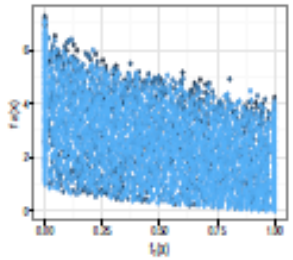
(i) WFG9

Multi-Objective Optimization

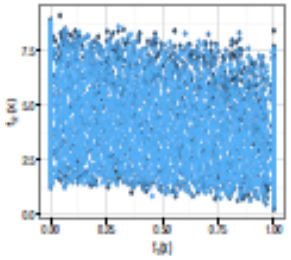
Vector-Evaluated PSO: Particle Movement for ZDT Problems



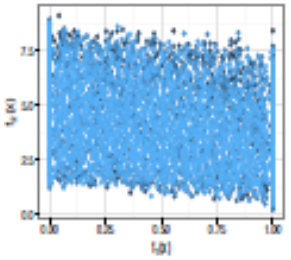
(a) ZDT1 S_1



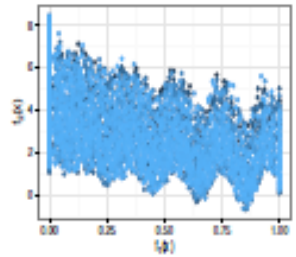
(b) ZDT1 S_2



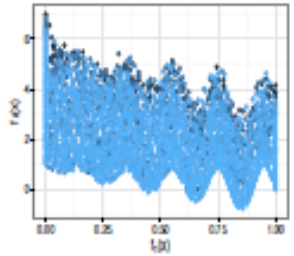
(c) ZDT2 S_1



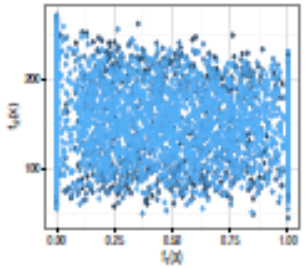
(c) ZDT2 S_1



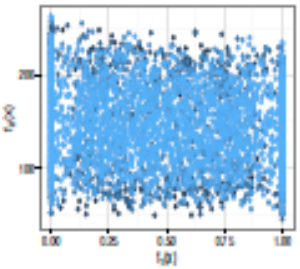
(e) ZDT3 S_1



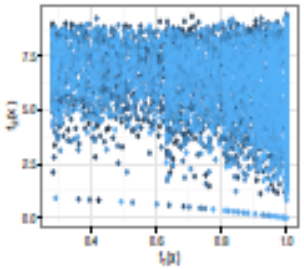
(f) ZDT3 S_2



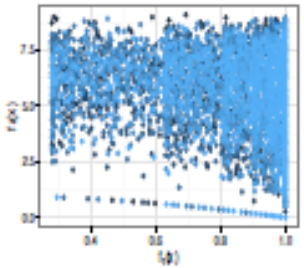
(g) ZDT4 S_1



(h) ZDT4 S_2



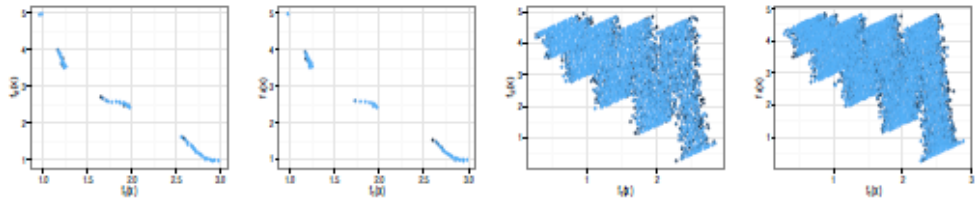
(i) ZDT6 S_1



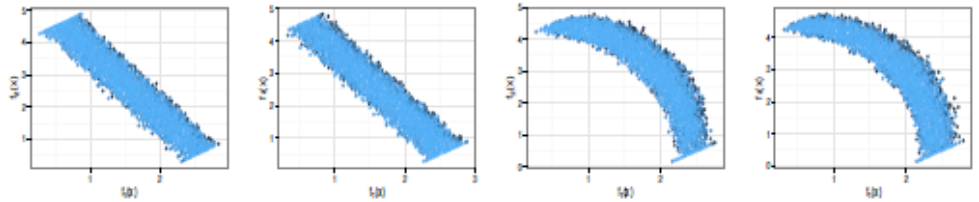
(j) ZDT6 S_2

Multi-Objective Optimization

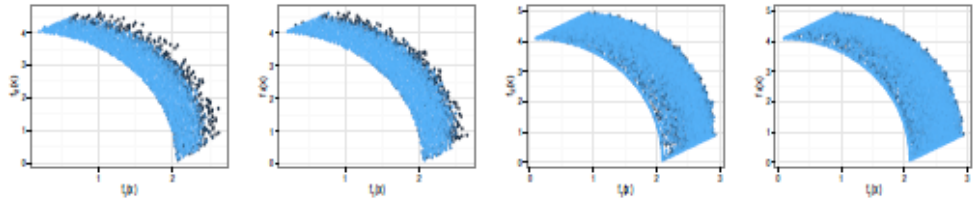
Vector-Evaluated PSO: Particle Movement for WFG Problems



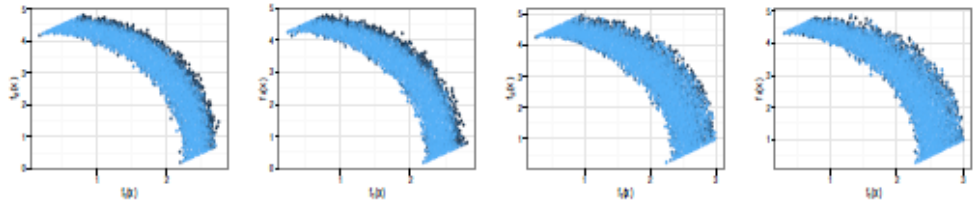
(a) WFG1 S_1 (b) WFG1 S_2 (c) WFG2 S_1 (d) WFG2 S_2



(e) WFG3 S_1 (f) WFG3 S_2 (g) WFG4 S_1 (h) WFG4 S_2



(i) WFG5 S_1 (j) WFG5 S_2 (k) WFG6 S_1 (l) WFG6 S_2



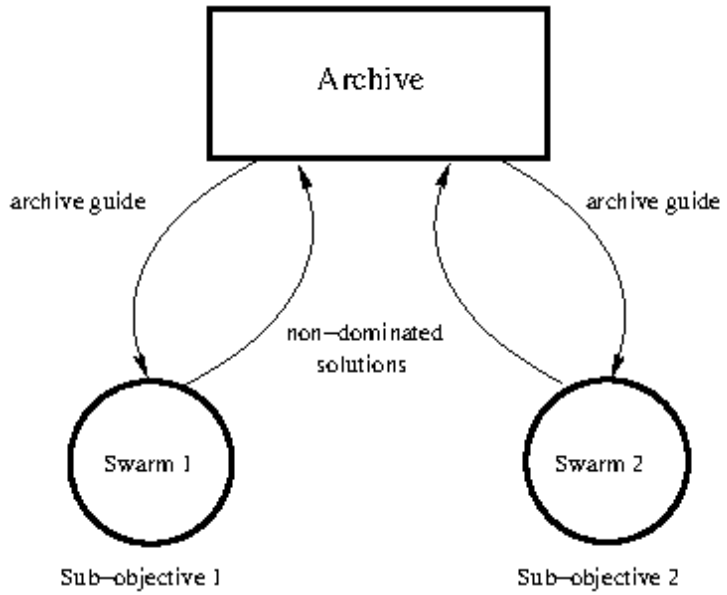
(m) WFG7 S_1 (n) WFG7 S_2 (o) WFG8 S_1 (p) WFG8 S_2

Multi-Objective Optimization

Multi-Guide Particle Swarm Optimization[25]

The multi-guide PSO (MGPSO) is a multi-swarm, cooperative PSO to solve multi-objective optimization problems

- One sub-swarm per sub-objective
- Local (pbest) & global (gbest) updates do not use a dominance relation, only the corresponding sub-objective function
- Maintains an external archive of non-dominated solutions
- Adds to the velocity update equation an archive guide
- The archive guide facilitates information exchange between sub-swarms about non-dominated solutions



[25] C Scheepers, AP Engelbrecht, CW Cleghorn, *Multi-Guide Particle Swarm Optimization for Multi-Objective Optimization: Enpirical and Stability Analysis*, 13(3-4):245–276, Swarm Intelligence, 2019

The velocity update:

$$\begin{aligned} v_{ij}(t+1) &= wv_{ij}(t) + c_1 r_{1j}(t)[y_{ij}(t) - x_{ij}(t)] \\ &+ \lambda_i c_2 r_{2j}(t)[\hat{y}_{ij}(t) - x_{ij}(t)] \\ &+ (1-\lambda_i) c_3 \mathbf{r}_r(\hat{a}_{ij}(t) - \mathbf{x}_{ij}(t)) \end{aligned}$$

- $\hat{\mathbf{a}}_i$ is a non-dominated solution in the archive
- λ_i controls the influence that the archine guide has:
  - Small values increase archive guide influence, decreasing influence of nbest
  - Large values decrease archive guide influence, increase influence of nbest
  - $\lambda_i \sim U(0,1)$
- $r_{3j} \sim U(0,1)$

The archive guide is selected randomly, as follows:

- Tournament selection is used, with competition pool randomly selected from the archive
- Select the non-dominated solution from the competition pool with the largest crowding distance
- Guides the MGPSO to focus more on sparsely populated areas of objective space

Archive is bounded:

- When full, an archive solution is removed from the denser areas of objective space

MGPSO algorithm initialization:

1: **for** each objective $m = 1, ..., n_m$ **do**
2: Let f_m be the objective function;
3: ▷ Create and initialize a swarm, S_m, to contain $S.n_{s_m}$ particles
4: **for** each particle $i = 1, \ldots, S.n_{s_m}$ **do**
5: Initialize position $S_m.\mathbf{x}_i(0)$ uniformly within a predefined hypercube of dimension n_x;
6: Initialize the personal best position as $S_m.\mathbf{y}_i(0) = S_m.\mathbf{x}_i(0)$;
7: Determine the neighborhood best position, $S_m.\hat{\mathbf{y}}_i(0)$;
8: Initialize the velocity as $S_m.\mathbf{v}_i(0) = \mathbf{0}$;
9: Initialize $S_m.\lambda_i \sim U(0, 1)$;
10: **end for**
11: **end for**
12: Let $t = 0$;

Multi-Guide Particle Swarm Optimization

The Algorithm (cont)

13: **repeat**
14: **for** each objective $m = 1, ..., n_m$ **do**
15: **for** each particle $i = 1, ..., S_m.n_s$ **do**
16: **if** $f_m(S_m.\mathbf{x}_i) < f_m(S_m.\mathbf{y}_i)$ **then**
17: $S_m.\mathbf{y}_i = S_m.\mathbf{x}_i(t)$;
18: **end if**
19: **for** particles $\hat{\imath}$ with particle i in their neighborhood **do**
20: **if** $f_m(S_m.\mathbf{y}_i) < f_m(S_m.\hat{\mathbf{y}}_{\hat{\imath}})$ **then**
21: $S_m.\hat{\mathbf{y}}_{\hat{\imath}} = S_m.\mathbf{y}_i$;
22: **end if**
23: **end for**
24: Update the archive with the solution $S_m.\mathbf{x}_i$;
25: **end for**
26: **end for**
27: **for** each objective $m = 1, ..., n_m$ **do**
28: **for** each particle $i = 1, ..., S_m.n_s$ **do**
29: Select a solution, $S_m.\hat{\mathbf{a}}_i(t)$, from the archive using tournament selection;
30: $$S_m.\mathbf{v}_i(t+1) = wS_m.\mathbf{v}_i(t) + c_1\mathbf{r}_1(S_m.\mathbf{y}_i(t) - S_m.\mathbf{x}_i(t)) + S_m.\lambda_i c_2\mathbf{r}_2(S_m.\hat{\mathbf{y}}_i(t) - S_m.\mathbf{x}_i(t)) + (1 - S_m.\lambda_i)c_3\mathbf{r}_3(S_m.\hat{\mathbf{a}}_i(t) - S_m.\mathbf{x}_i(t)));$$
31: $S_m.\mathbf{x}_i(t+1) = S_m.\mathbf{x}_i(t) + S_m.\mathbf{v}_i(t+1)$;
32: **end for**
33: **end for**
34: $t = t + 1$;
35: **until** stopping condition is true

ZDT (2-objectives) and WFG (2- and 3-objectives)

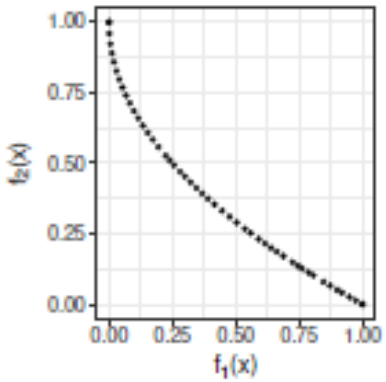
Name	Separability	Modality	Geometry
ZDT1	separable	unimodal	convex
ZDT2	separable	unimodal	concave
ZDT3	separable	unimodal/multimodal	disconnected
ZDT4	separable	unimodal/multimodal	convex
ZDT6	separable	multimodal	concave
WFG1	separable	unimodal	convex, mixed
WFG2	non-separable	unimodal/multimodal	convex, disconnected
WFG3	non-separable	unimodal	linear, degenerate
WFG4	separable	multimodal	concave
WFG5	separable	multimodal	concave
WFG6	non-separable	unimodal	concave
WFG7	separable	unimodal	concave
WFG8	non-separable	unimodal	concave
WFG9	non-separable	multimodal, deceptive	concave

- Total number of particles of 50, divided among the subswarms
- w, c_1, c_2 and c_3 tuned on each problem
- The distribution of the particles among the swarms also tuned

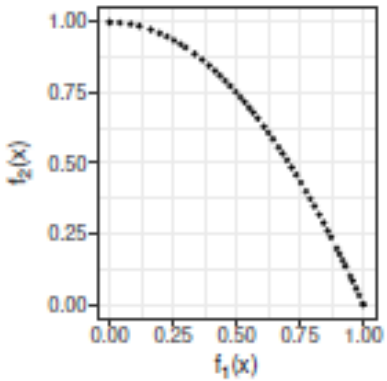
Problem	Objectives	$\|S_1\|$	$\|S_2\|$	$\|S_3\|$	T	w	c_1	c_2	c_3
ZDT1	2	33	17		3	0.475	1.80	1.10	1.80
ZDT2	2	8	42		3	0.075	1.60	1.35	1.90
ZDT3	2	8	42		3	0.050	1.85	1.90	1.90
ZDT4	2	5	45		2	0.175	1.85	1.35	1.85
ZDT6	2	1	49		3	0.600	1.85	1.55	1.80
WFG1	2	45	5		3	0.275	1.65	1.80	1.75
WFG2	2	24	26		2	0.750	1.15	1.70	1.05
WFG3	2	31	19		2	0.600	1.60	1.85	0.95
WFG4	2	2	48		2	0.100	0.80	1.65	1.70
WFG5	2	50	0		2	0.600	0.80	1.60	1.85
WFG6	2	19	31		2	0.525	0.65	0.60	1.65
WFG7	2	29	21		2	0.450	1.20	1.85	1.55
WFG8	2	37	13		3	0.750	1.00	1.65	1.05
WFG9	2	13	37		2	0.275	1.00	0.50	1.70
WFG1	3	37	4	9	2	0.125	1.20	1.30	1.75
WFG2	3	24	25	1	2	0.275	1.25	1.40	1.70
WFG3	3	29	10	11	2	0.525	1.65	1.75	0.75
WFG4	3	29	21	0	2	0.275	1.75	0.50	1.05
WFG5	3	2	48	0	3	0.575	0.60	1.85	1.75
WFG6	3	5	30	15	3	0.300	0.90	0.90	1.90
WFG7	3	10	22	18	2	0.425	1.45	1.50	1.40
WFG8	3	4	23	23	3	0.425	0.95	1.75	1.85
WFG9	3	4	45	1	2	0.275	1.25	0.75	1.50

Multi-Objective Optimization

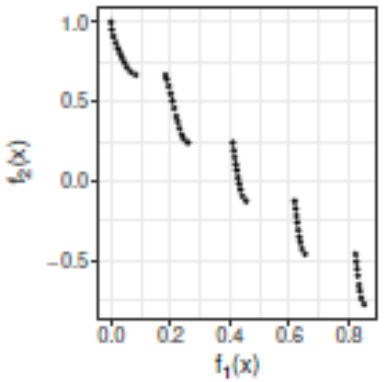
Multi-Guide Particle Swarm Optimization: Obtained POFs: ZDT Problems



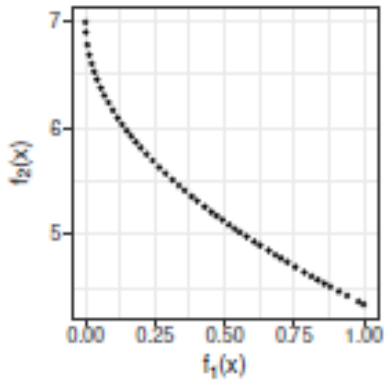
(a) ZDT1



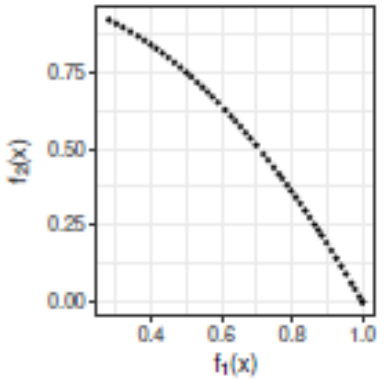
(b) ZDT2



(c) ZDT3



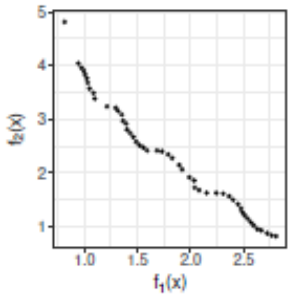
(d) ZDT4



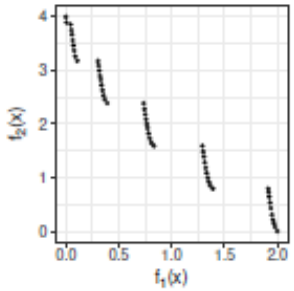
(e) ZDT6

Multi-Objective Optimization

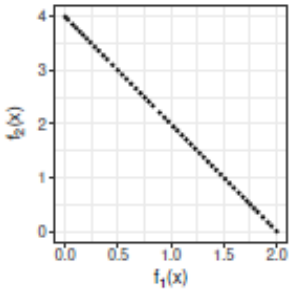
Multi-Guide Particle Swarm Optimization: Obtained POFs: WFG Problems



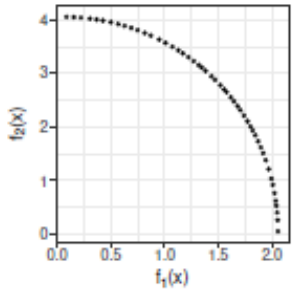
(a) WFG1



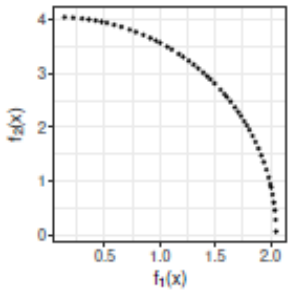
(b) WFG2



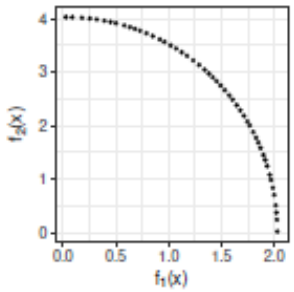
(c) WFG3



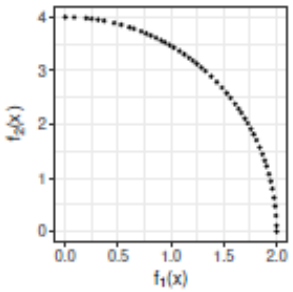
(d) WFG4



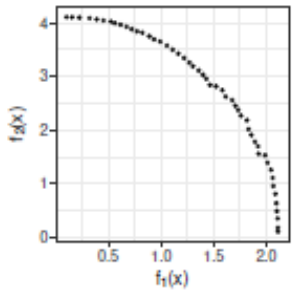
(e) WFG5



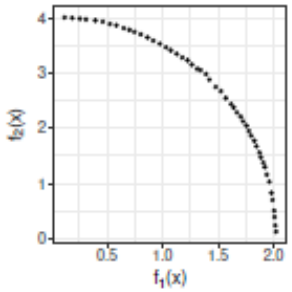
(f) WFG6



(g) WFG7



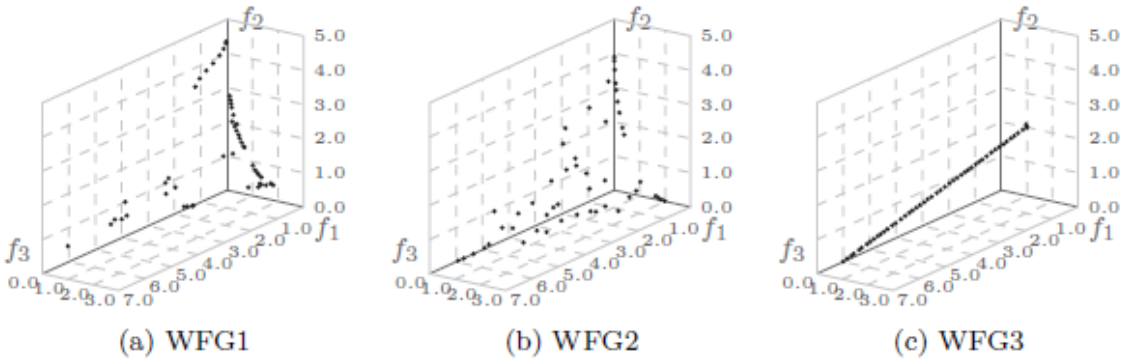
(h) WFG8



(i) WFG9

Multi-Objective Optimization

Multi-Guide Particle Swarm Optimization: Obtained POFs: WFG Problems



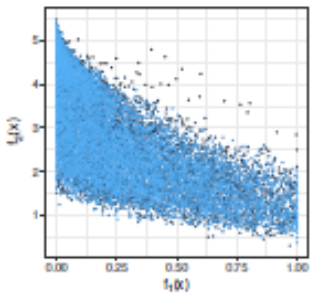
(a) WFG1 (b) WFG2 (c) WFG3

(d) WFG4 (e) WFG5 (f) WFG6

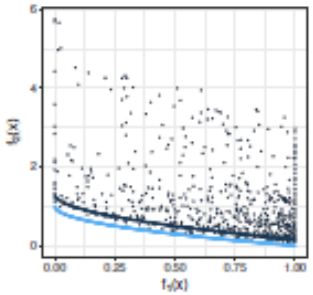
(g) WFG7 (h) WFG8 (i) WFG9

Multi-Objective Optimization

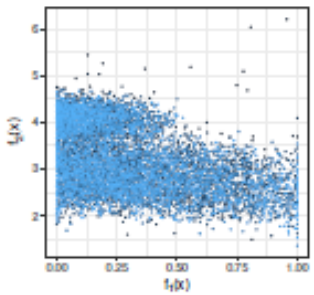
Multi-Guide Particle Swarm Optimization: Candidate Solutions Movement



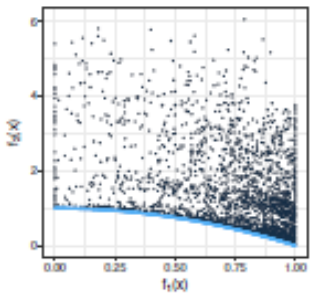
(a) ZDT1 S_1



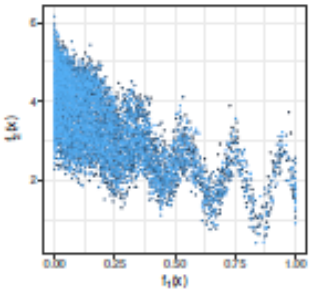
(b) ZDT1 S_2



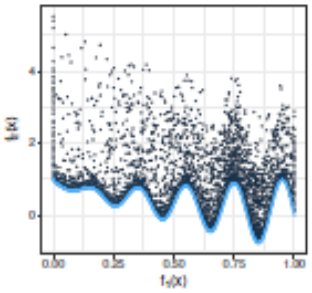
(c) ZDT2 S_1



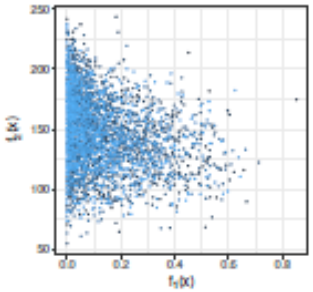
(d) ZDT2 S_2



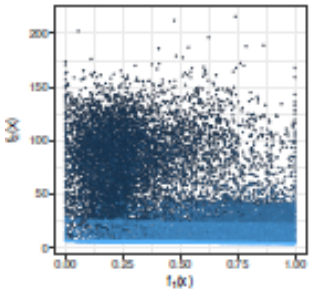
(e) ZDT3 S_1



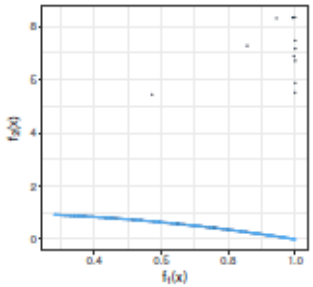
(f) ZDT3 S_2



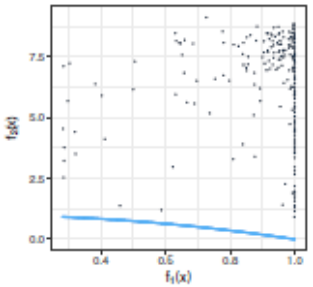
(g) ZDT4 S_1



(h) ZDT4 S_2



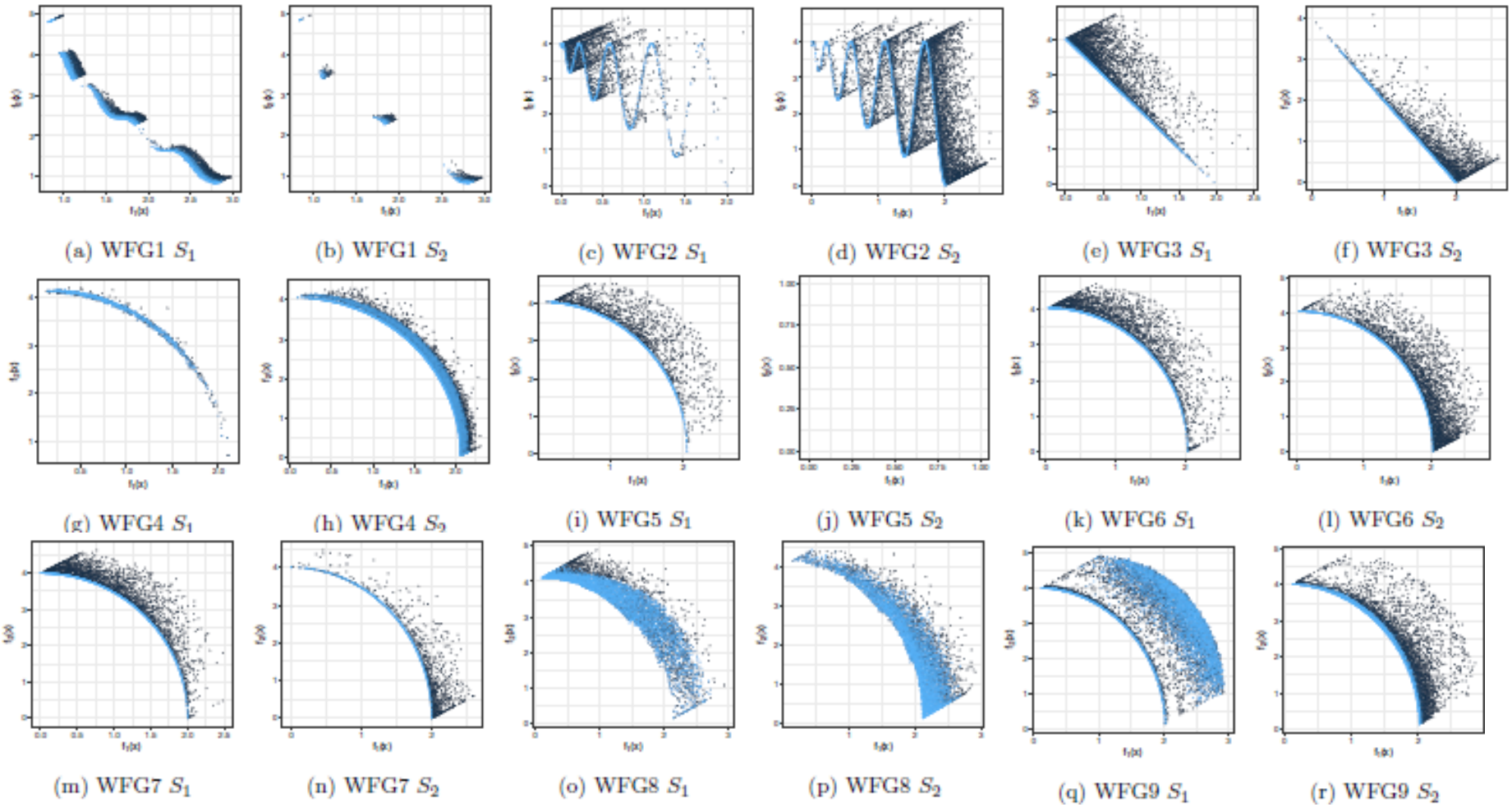
(i) ZDT6 S_1



(j) ZDT6 S_2

Multi-Objective Optimization

Multi-Guide Particle Swarm Optimization: Candidate Solutions Movement (cont)



(a) WFG1 S_1 (b) WFG1 S_2 (c) WFG2 S_1 (d) WFG2 S_2 (e) WFG3 S_1 (f) WFG3 S_2

(g) WFG4 S_1 (h) WFG4 S_9 (i) WFG5 S_1 (j) WFG5 S_2 (k) WFG6 S_1 (l) WFG6 S_2

(m) WFG7 S_1 (n) WFG7 S_2 (o) WFG8 S_1 (p) WFG8 S_2 (q) WFG9 S_1 (r) WFG9 S_2

Compared with other MOPSO algorithms

Algorithm	Result	ZDT Function 1	2	3	4	6	Overall
MGPSO	Wins	2	3	3	3	3	14
	Losses	1	1	1	1	1	5
	Difference	1	2	2	2	2	9
	Rank	2	2	2	2	2	2
SMPSO	Wins	4	4	4	4	4	20
	Losses	0	0	0	0	0	0
	Difference	4	4	4	4	4	20
	Rank	**1**	**1**	**1**	**1**	**1**	**1**
OMOPSO	Wins	1	2	1	2	0	6
	Losses	3	2	3	2	3	13
	Difference	-2	0	-2	0	-3	-7
	Rank	4	3	4	3	4	4
$\text{VEPSO}_{\text{Random}}$	Wins	0	1	0	0	0	1
	Losses	4	3	4	4	3	18
	Difference	-4	-2	-4	-4	-3	-17
	Rank	5	4	5	5	4	5
$\text{VEPSO}_{\text{PCXA}}$	Wins	2	0	2	1	2	7
	Losses	1	4	2	3	2	12
	Difference	1	-4	0	-2	0	-5
	Rank	2	5	3	4	3	3

Multi-Objective Optimization

Multi-Guide Particle Swarm Optimization: Inverted Generational Distance (cont)

Algorithm	Result	2-objective WFG Function									Overall
		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	
MGPSO	Wins	4	4	4	2	3	3	4	3	3	30
	Losses	0	0	0	2	1	0	0	0	1	4
	Difference	4	4	4	0	2	3	4	3	2	26
	Rank	**1**	**1**	**1**	3	2	**1**	**1**	**1**	2	**1**
SMPSO	Wins	2	2	2	4	4	3	2	2	4	25
	Losses	2	1	2	0	0	0	2	2	0	9
	Difference	0	1	0	4	4	3	0	0	4	16
	Rank	3	2	3	**1**	**1**	**1**	3	3	**1**	2
OMOPSO	Wins	3	2	3	3	0	0	3	3	0	17
	Losses	1	1	1	1	4	3	1	0	4	16
	Difference	2	1	2	2	-4	-3	2	3	-4	1
	Rank	2	2	2	2	5	4	2	**1**	5	3
$\text{VEPSO}_{\text{Random}}$	Wins	0	0	0	0	1	2	0	0	2	5
	Losses	4	4	4	4	3	2	4	4	2	31
	Difference	-4	-4	-4	-4	-2	0	-4	-4	0	-26
	Rank	5	5	5	5	4	3	5	5	3	5
$\text{VEPSO}_{\text{PCXA}}$	Wins	1	1	1	1	2	0	1	1	1	9
	Losses	3	3	3	3	2	3	3	3	3	26
	Difference	-2	-2	-2	-2	0	-3	-2	-2	-2	-17
	Rank	4	4	4	4	3	4	4	4	4	4

Multi-Objective Optimization

Multi-Guide Particle Swarm Optimization: Inverted Generational Distance (cont)

Algorithm	Result	3-objective WFG Function									Overall
		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	
MGPSO	Wins	4	4	4	4	4	3	4	4	4	35
	Losses	0	0	0	0	0	0	0	0	0	0
	Difference	4	4	4	4	4	3	4	4	4	35
	Rank	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**
SMPSO	Wins	3	2	2	3	2	3	3	2	3	23
	Losses	1	2	2	1	2	0	1	2	1	12
	Difference	2	0	0	2	0	3	2	0	2	11
	Rank	2	3	3	2	3	**1**	2	3	2	2
OMOPSO	Wins	2	3	3	2	0	0	2	3	0	15
	Losses	2	1	1	2	4	3	2	1	4	20
	Difference	0	2	2	0	-4	-3	0	2	-4	-5
	Rank	3	2	2	3	5	4	3	2	5	3
$\text{VEPSO}_{\text{Random}}$	Wins	0	1	1	1	3	0	1	1	1	9
	Losses	3	3	3	3	1	3	3	3	2	24
	Difference	-3	-2	-2	-2	2	-3	-2	-2	-1	-15
	Rank	4	4	4	4	2	4	4	4	3	4
$\text{VEPSO}_{\text{PCXA}}$	Wins	0	0	0	0	1	2	0	0	1	4
	Losses	3	4	4	4	3	2	4	4	2	30
	Difference	-3	-4	-4	-4	-2	0	-4	-4	-1	-26
	Rank	4	5	5	5	4	3	5	5	3	5

Compared with MOEAs

Algorithm	Result	ZDT Function 1	2	3	4	6	Overall
MGPSO	Wins	4	4	4	0	3	15
	Losses	0	0	0	4	0	4
	Difference	4	4	4	-4	3	11
	Rank	**1**	**1**	**1**	5	**1**	**1**
NSGA II	Wins	2	2	3	3	2	12
	Losses	2	2	1	0	2	7
	Difference	0	0	2	3	0	5
	Rank	3	3	2	**1**	3	3
MOEA/D	Wins	3	3	0	3	3	12
	Losses	1	1	3	0	0	5
	Difference	2	2	-3	3	3	7
	Rank	2	2	5	**1**	**1**	2
SPEA2	Wins	0	0	1	2	0	3
	Losses	3	4	2	2	4	15
	Difference	-3	-4	-1	0	-4	-12
	Rank	4	5	3	3	5	4
PESA-II	Wins	0	1	0	1	1	3
	Losses	3	3	2	3	3	14
	Difference	-3	-2	-2	-2	-2	-11
	Rank	4	4	4	4	4	4