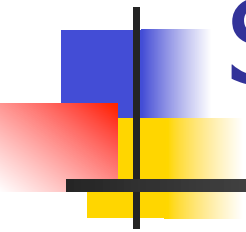
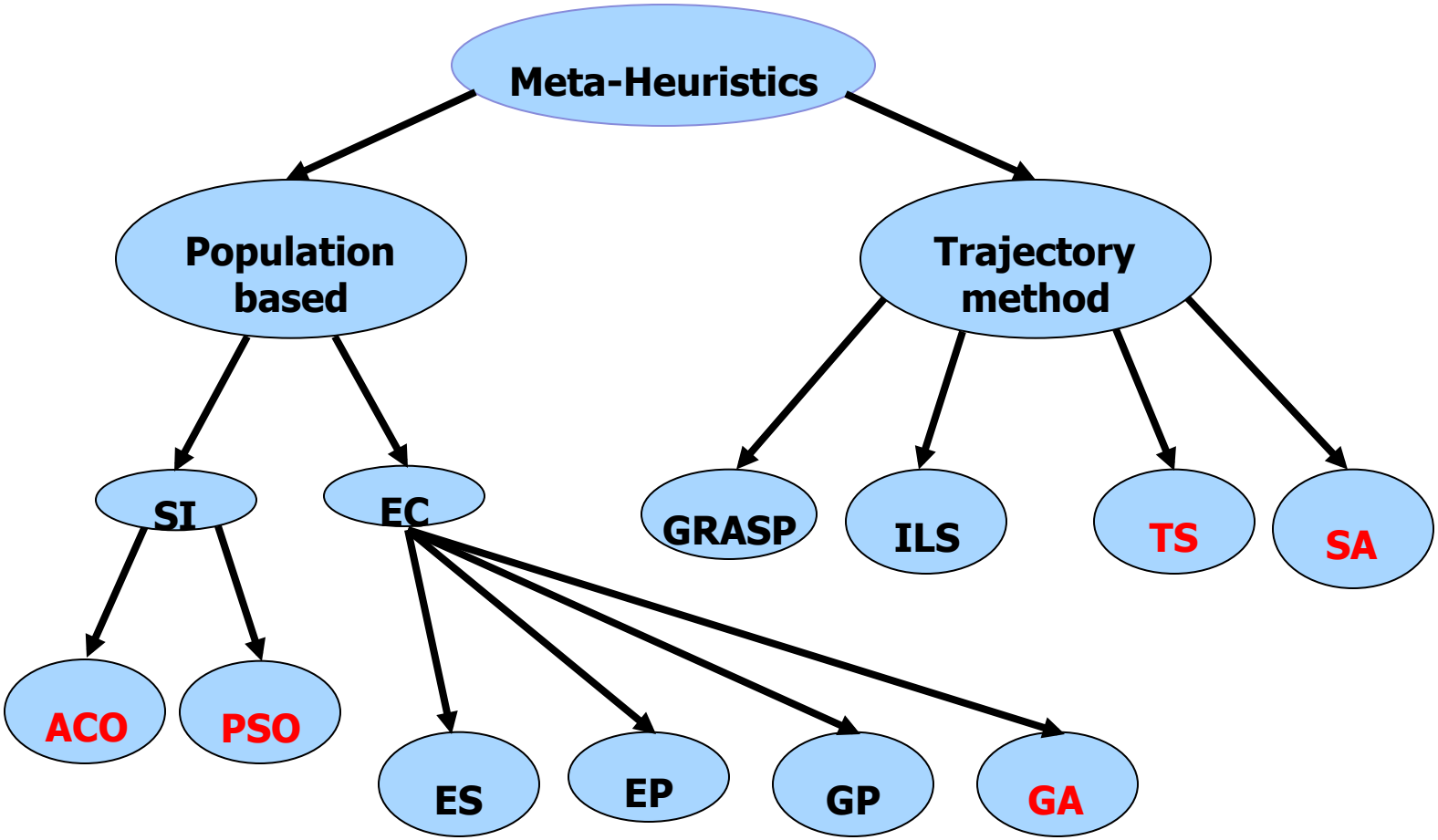


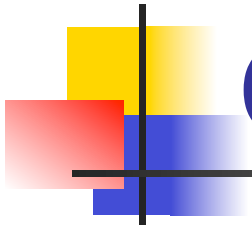
Cooperative and Adaptive Search Algorithms

Swarm Intelligence

(Based on material from the Text and some papers listed in the last slide)

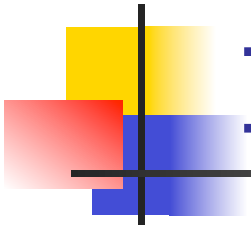
Meta-Heuristics





Outline

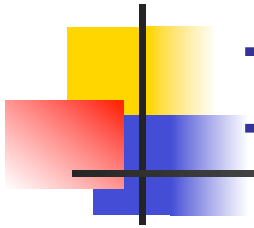
- Introduction,
- Examples from Nature,
- Properties,
- Models of behaviour,
- How is a tool designed?
- A successful SI tool,
- References.



Introduction

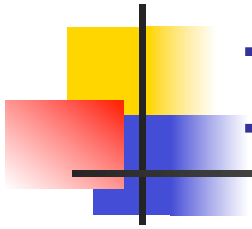
- **Swarm Intelligence:** A recent computational metaphor that is inspired by the collective behaviour of social insects, vertebrates, animals, bacteria.

- **Swarm:** A group of agents which communicate with each others by acting on their local environment.



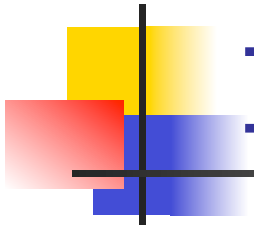
Introduction

- **The interaction between the agents result in distributive collective problem solving strategy.**
- **This complex behavior is not a property of any single individual (or agent) of the swarm but rather *emerges* from their interaction.**



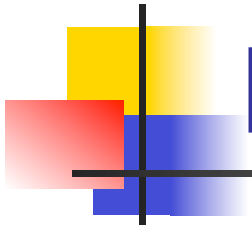
Introduction

- Interaction in biological systems happens in a number of of ways:
  - Direct (physical contact, or by visual, audio or chemical perceptual inputs),
  - Indirect (via local change to the environment), *Stigmergy*



Introduction

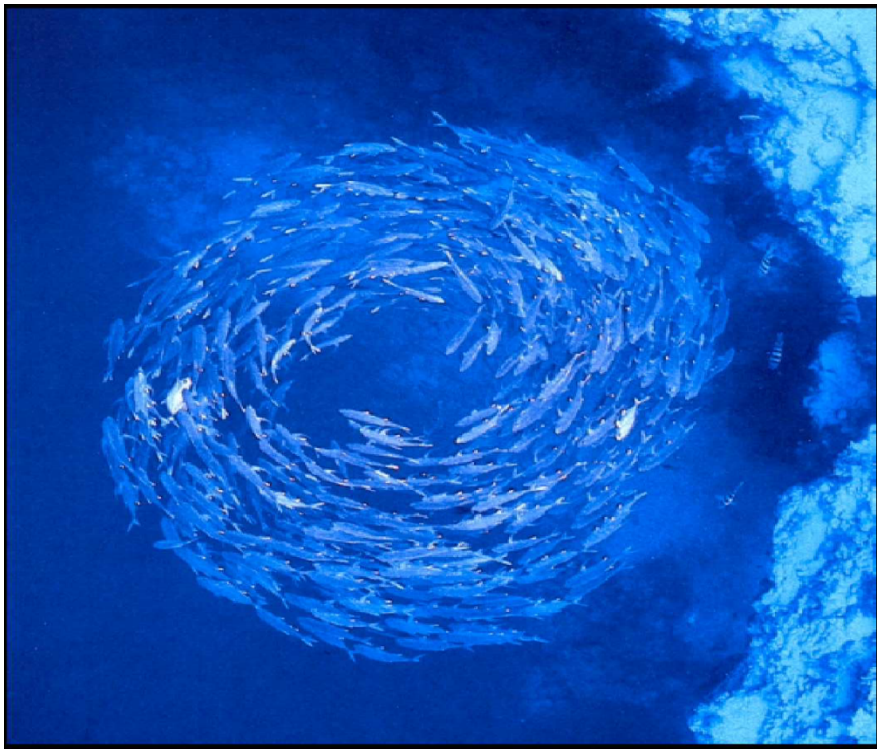
- The word *Stigmergy* is due to Pierre-Paul Grasse who was studying the behavior of termites,
- He observed that the insects react to signals that activate a genetically encoded reaction,
- The effect of these reactions serve as new signals for both:
  - The insect that produced them,
  - Other insects in the colony.



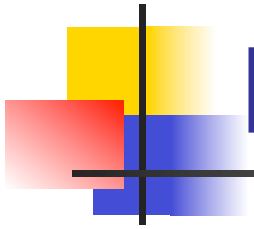
Examples From Nature

- Examples of emergent behavior from nature include:
  - Swimming direction in fish schooling,
  - Spatial patterns in bird flocking,
  - Agricultural behaviours,
  - Nest building by termites, bees, ants
  - Dancing in bee foraging

Examples from Nature



Fish schooling (© CORO, CalTech)



Examples From Nature

-

Bird flocking – V formation (© Soren Breitling)

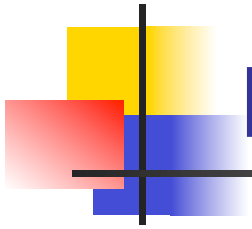
Examples from Nature

Ants' agricultural behaviour (© Alex Wild)

Examples from Nature

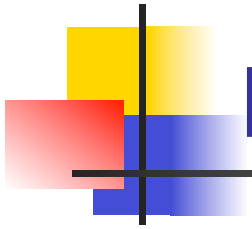
.

Wasps’ nest building (© Dominique Snyers)



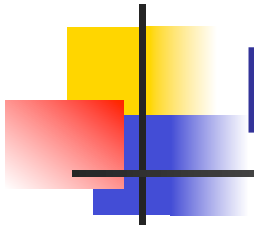
Examples of Emergent Behavior

- Emergent behavior has been observed in other non-biological systems.
- Stock market is a good example of collective behavior.
- Each investor has limited knowledge and affects the market in a limited way
- Interactions create a market behavior



Examples of Emergent Behavior

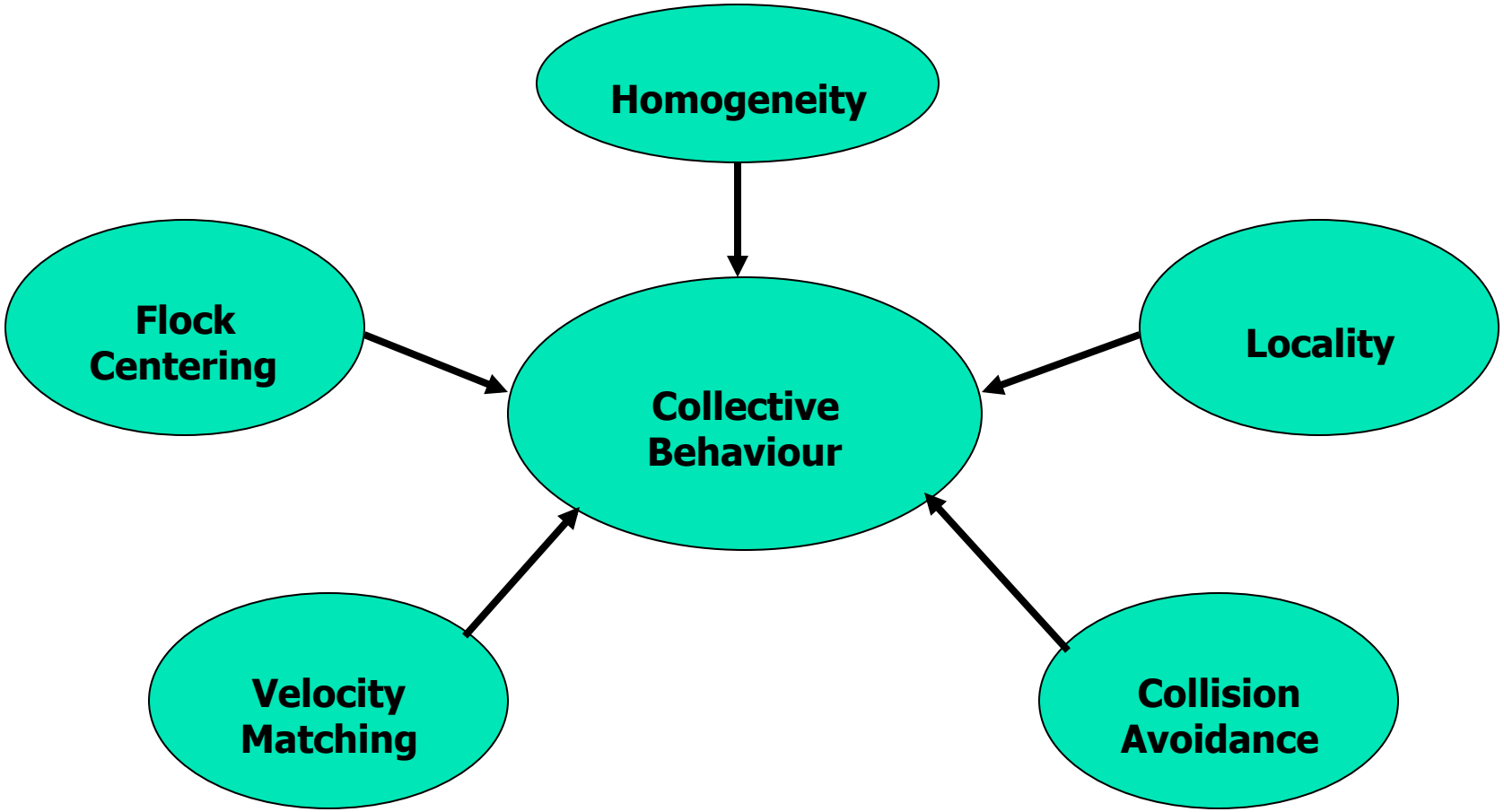
- Another example is traffic patterns and self organizing behavior that results in traffic.
- A third example is the spatial structure of galaxies is an emergent property of distribution of energy and matter in the universe.

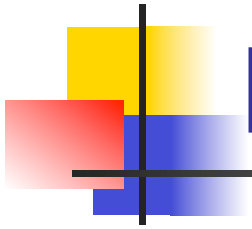


Properties

- These behaviour were inspiring as they have some interesting properties:
  - **Flexibility:** the system performance is adaptive with respect to internal or external changes,
  - **Robustness:** the system always perform even if some individuals fail,
  - **Decentralization:** the control is distributed among the individuals,
  - **Self-organization:** global behaviours appear out of local individual interactions.

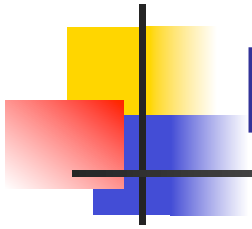
Properties (Flocking)





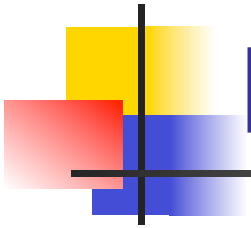
Properties (Flocking)

- **Homogeneity:** all the entities are similar, the flock moves without a leader (decentralized control)
- **Locality:** any entity communicates with nearby entities only,
- **Collision avoidance:** avoid colliding with nearby entities,
- **Velocity matching:** attempt to match velocity of nearby entities,
- **Flock centering:** attempt to stay close to nearby entities.



Properties (Flocking)

- Usually, the attempt to stay close to nearby entities has the highest priority,
- This is because every entity mainly tries not to be isolated.



Models of behaviour

- Couzin et. al [1], identified different behavioural models:
  - Swarm,
  - Torus,
  - Dynamic parallel group,
  - Highly parallel group.

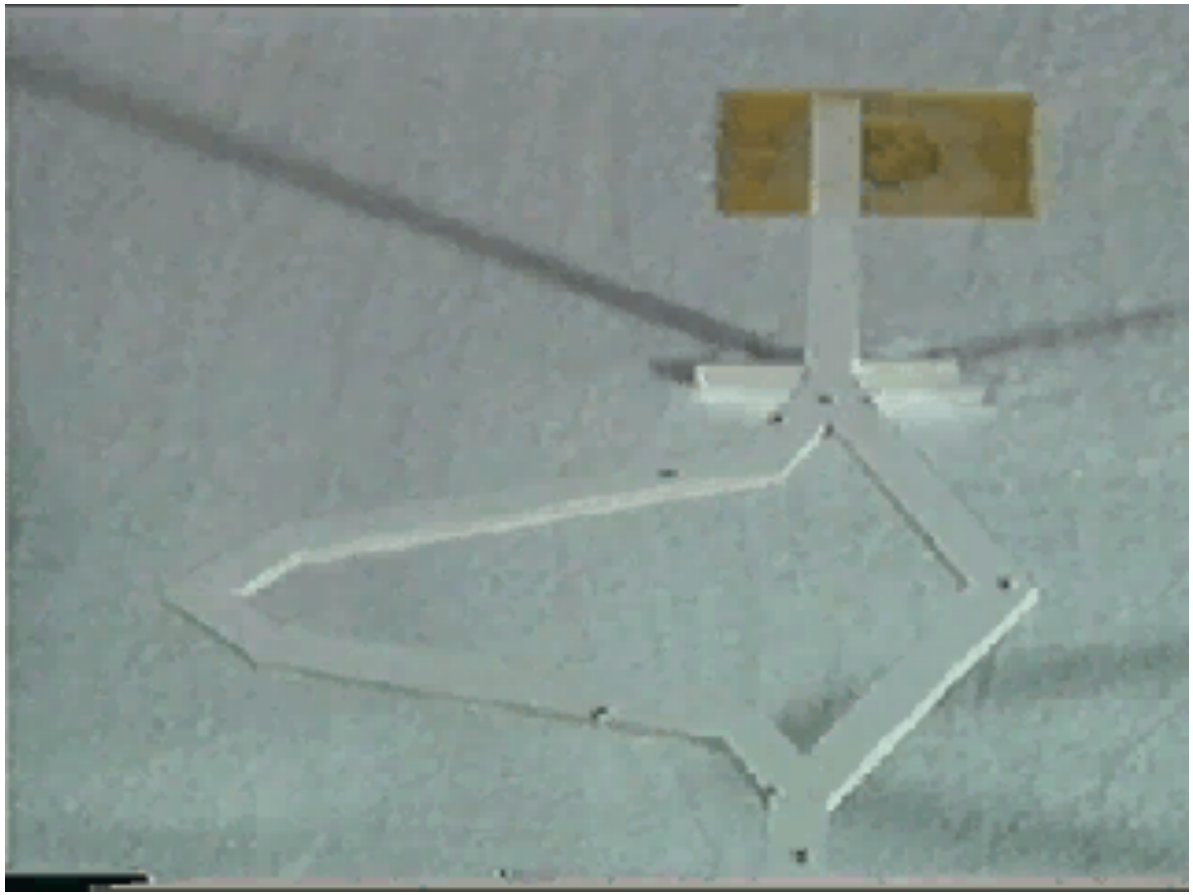
Models of behaviour

From [2]

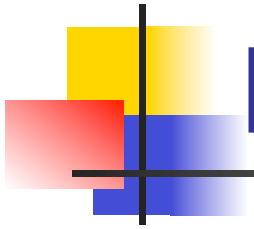
.

Swarm
Coherent group with low level of polarization (parallel alignment),

Introduction



http://staff.washington.edu/paymana/swarm/videos/ants_on_bridge.avi



Models of behaviour

From [2]

.

Torus
Entities rotate around an empty core with a random direction of rotation

Models of behaviour

From [2]

.

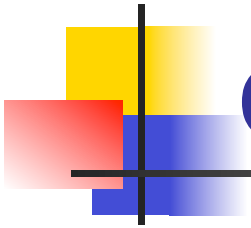
Dynamic parallel group
Entities are polarized and move as a coherent group, but they can move through out the group and density and group form fluctuates

Models of behaviour

From [2]

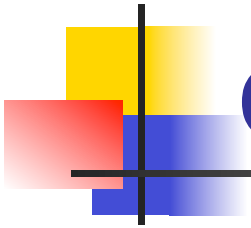
.

Highly parallel group
Same as previous one but with minimum fluctuations



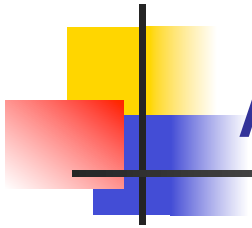
Computational Swarm Intelligence

- Building computational models to solve complex problems based on models of behaviors in biological swarms.
- Design issues involve: modeling the agent (or individual), the interaction process, adaptation and cooperation.



Computational Swarm Intelligence

- Ant Colony Optimization (**ACO**) and Particle Swarm Optimization (**PSO**) are two examples of computational swarm intelligence models.
- ACO models the very simple behavior of of pheromone trail following of ants.
- PSO models 2 simple behaviors: moving towards its best closest neighbor and moving back to its own best state.

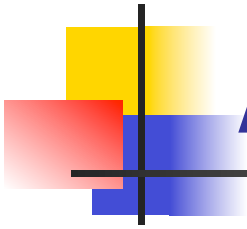


ACO and PSO

- Communication is different in each. PSO uses neighbors broadcast (direct communication), ACO uses *stigmergy* through pheromone perception.
- The agent in both is very simple and doesn’t learn at the individual level.

A successful CSI tool

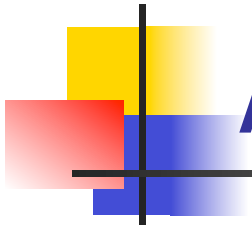
- There are five basic principles that should be present [3]:
  - Proximity,
  - Quality,
  - Diverse response,
  - Stability,
  - Adaptability.



A successful SI tool

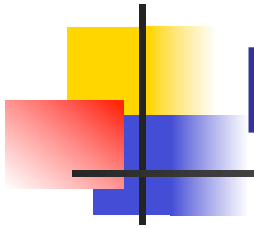
- **Proximity:** the swarm should be able to carry out simple space and time computations,

- **Quality:** the swarm should be able to respond to quality factors in the environment.



A successful SI tool

- **Diverse Response:** the swarm should not commit its activities along excessively narrow channels,
- **Stability:** the swarm should not change its mode of behaviour every time the environment changes,
- **Adaptability:** the swarm must be able to change behaviour mode when it is worth the computational price.

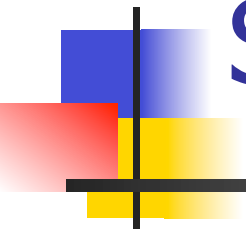


References

Text: Fundamentals of Computational Swarm Intelligence by A. P. Engelbrecht, Wiely, 2005. Chapter 1.

Papers:

- ID. Couzin, J. Krause, R. James, GD. Ruxton, NR. Franks.”Collective Memory and Spatial Sorting in Animal Groups”. Journal of Theoretical Biology, 218, pp. 1–11, 2002.
- C. Grosan, A. Abraham and C. Monica.”Swarm Intelligence in Data Mining”. In Swarm Intelligence in Data Mining, A. Abraham, C. Grosan and V. Ramos (Eds), Springer, pp. 1–16, 2006..
- MM. Millonas .“Swarms, phase transitions, and collective intelligence”. In CG. Langton Ed., Artificial Life III, Addison Wesley, Reading, MA, 1994.

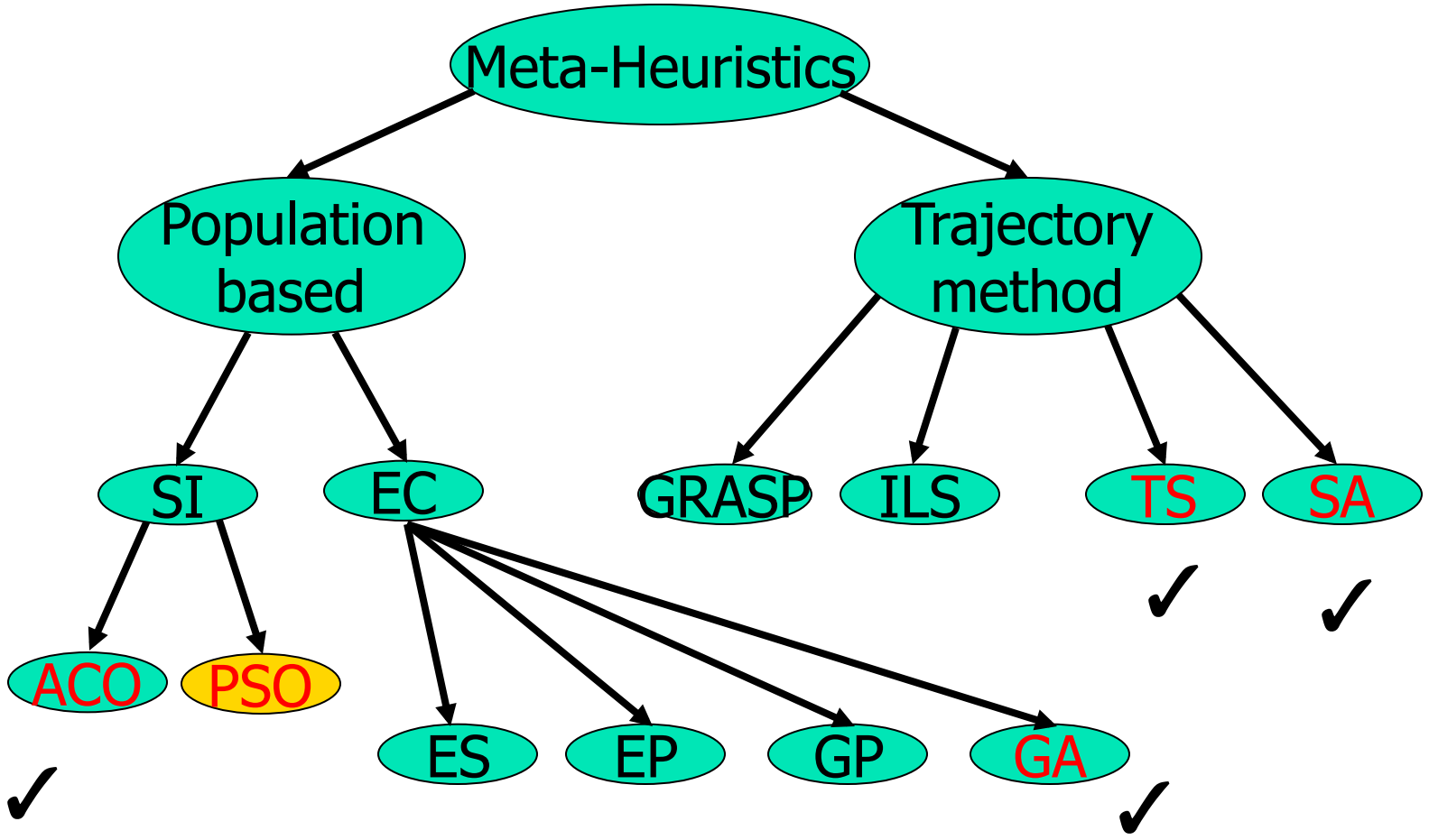


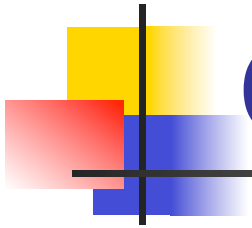
Cooperative and Adaptive Search Algorithms

Particle Swarm Optimization (PSO)

(Material based on text and references cited)

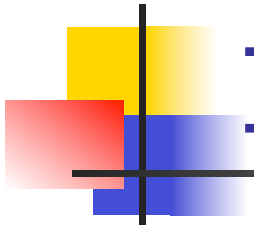
Meta-Heuristics





Outline

- Introduction,
- PSO,
- Discrete PSO,
  - Binary PSO
  - Permutation PSO,
- Applications,
- Cooperation,
- References.

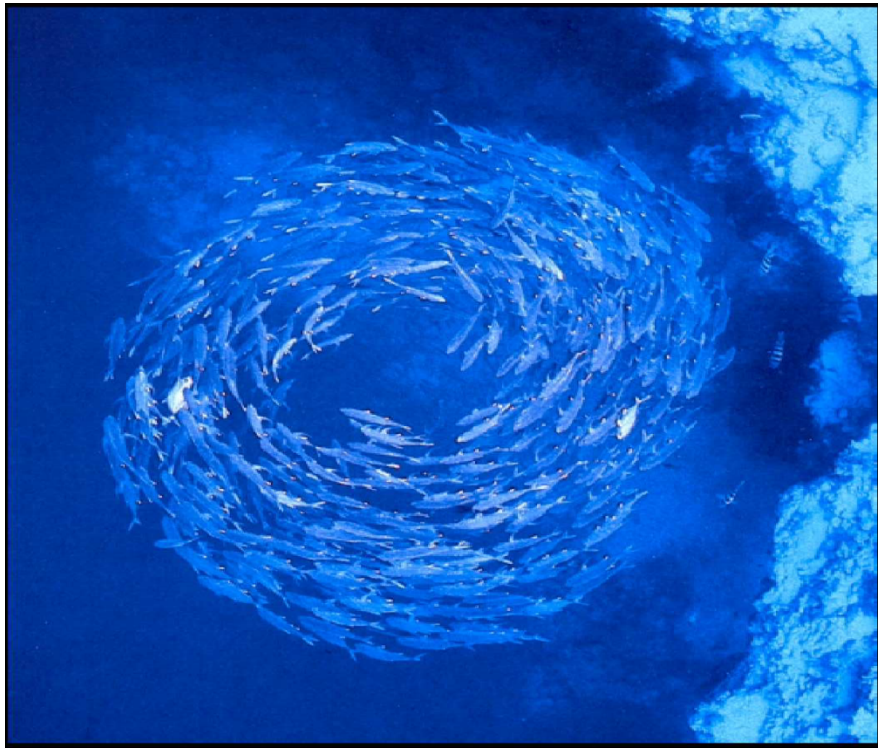


Introduction

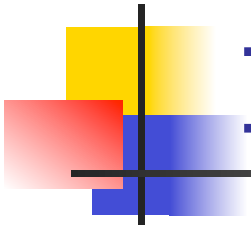
- The main idea is to simulate the collective behavior of social animals
- In particular, bird flocking and fish schooling behaviors
- Unlike some animal teams where there is a leader (e.g a pride of lions or a troop of baboons), the interest here in teams that has no leader
- Individuals have no knowledge of the global behavior of the group
- They have the ability to move together based on social interaction between neighbours

Introduction

Bird flocking – V formation (© Soren Breitling)



Fish schooling (© CORO, CalTech)



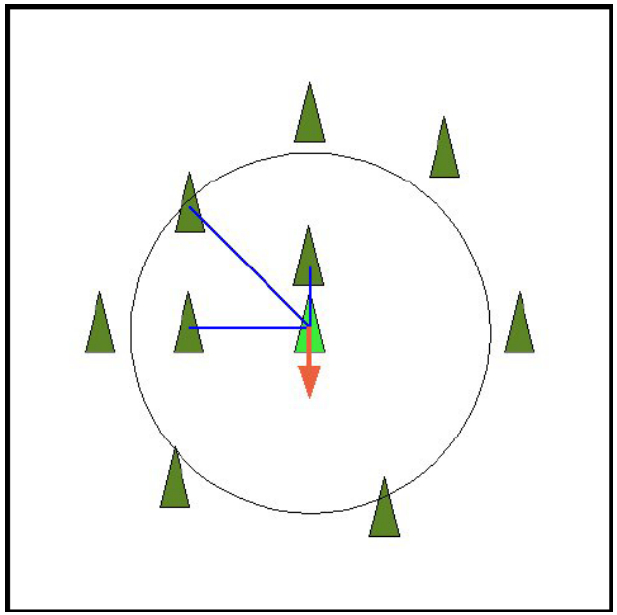
Introduction

- The first computer program was written by Reynolds in 1986 [1] to simulate swarms for computer graphics and movies,
- The work took account of three behaviours:
  - Separation,
  - Alignment,
  - Cohesion.
- For online simulations, refer to [2].

  http://www.red3d.com/cwr/boids/

Introduction

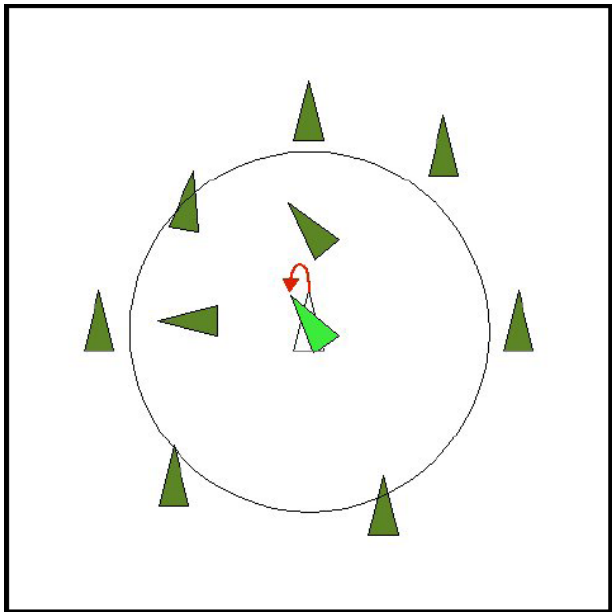
- Separation. Each agent tries to move away from its nearby mates if they are too close (Collision Avoidance).



From [3]

Introduction

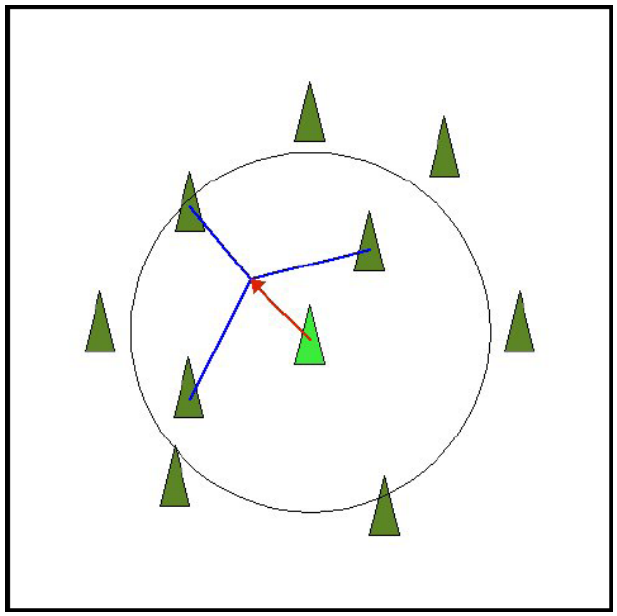
- Alignment. Each agent steers towards the average heading of its nearby mates (Velocity matching).



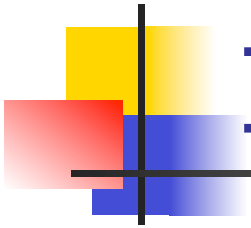
From [3]

Introduction

- Cohesion. Each agent tries to go towards the average position of its nearby mates (Centering or position control).

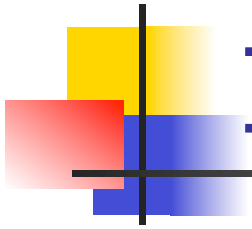


From [3]



Introduction

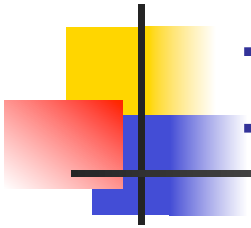
- Heppner and Grenander [4] used a similar flocking model but added a roost (place for birds to rest) as an attractor of the birds.
- The intent was to provide a computer simulation of a flock of birds to understand the underlying rules that enable synchronous flocking



Introduction

- Kennedy and Eberhart in 1995 [5, 6], introduced an optimization method based on the simple behavior of emulating the success of neighboring individuals.

- Followed the same steps taken by Reynolds and adding a *Roost* as proposed by Heppner and Grenander

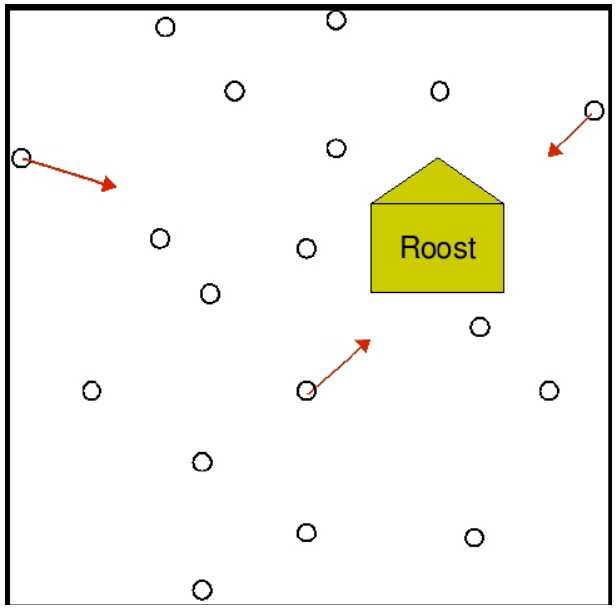


Introduction

- The *roost* is in the form of a memory of previous own best and neighborhood best positions (referred to *cornfield*)
- These two best positions serve as attractor
- By adjusting the positions of the flock proportion to the distance from the best positions, they converge to the goal

Introduction

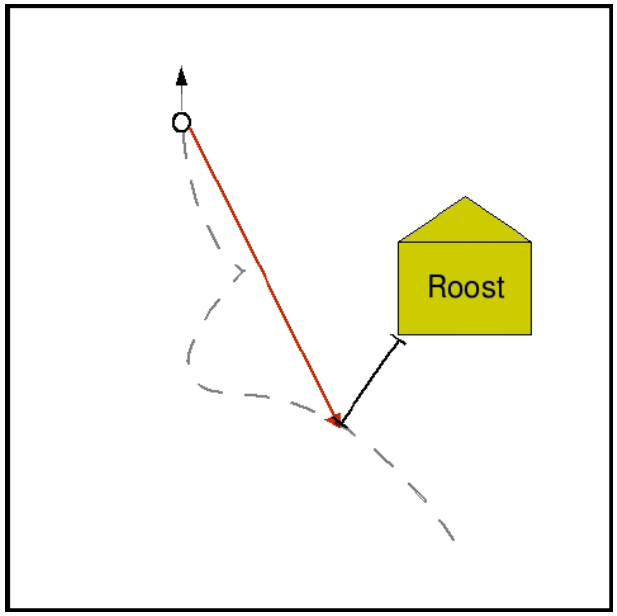
- All the individuals are attracted to the roost.



From [3]

Introduction

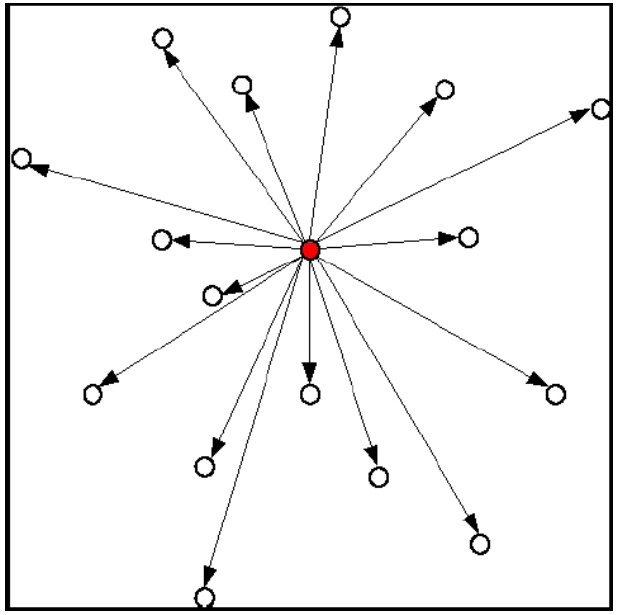
- Each memorizes the position in which it was closest to the roost.



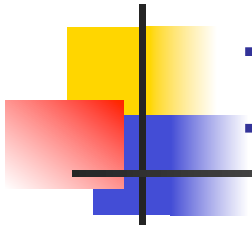
From [3]

Introduction

- Each shares its information with all the others.

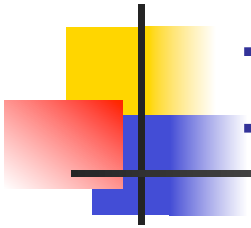


From [3]



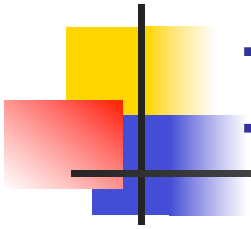
Introduction

- At the end of the simulation, all the individuals landed on the roost,
- It was realized, this could be used to solve optimization problems,
- If the distance to the roost was changed by some unknown function, the individuals land on the minimum



Introduction

- Kennedy and Eberhart called their model Particle Swarm Optimization (*PSO*)
- They choose the word *particle* to mean individual or candidate solution (in optimization terms) as they felt it is more appropriate for the the use with velocity and acceleration



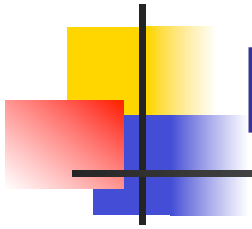
Introduction

- As their paradigm is a simplified version of bird flocking they preferred the use of the word *swarm* to indicate the population.
- PSO is a population based approach similar to GA and other EC approaches

Introduction

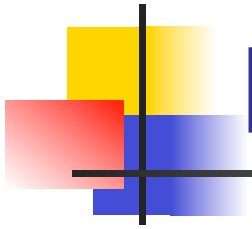
- In comparison with GA:

PSO		GA
Swarm	⟷	Population
Particle	⟷	Individual
Fitness	⟷	Fitness
Less fit don't die	⟷	Survival of the fittest
Uses past experience and relationship to neighbours	⟷	Uses crossover and mutations



PSO

- Introduction
- Motion,
- A simple algorithm,
- A different algorithm,
- Convergence,
- Neighborhoods.

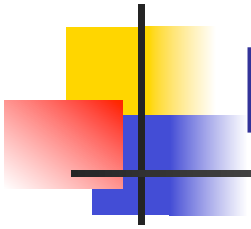


PSO - Introduction

- A stochastic optimization approach that manipulates a number of candidate solutions at once,
- A solution is referred to as a *particle*, the whole population is referred to as a *swarm*,
- Each particle holds information essential for its movement.

PSO - Motion

- Each particle holds:
  - Its current position x_i,
  - Its current velocity v_i,
  - The best position it achieved so far, personal best, $pbest_i$, or sometimes p_i for short,
  - The best position achieved by particles in its neighbourhood *Nbest*
  - If the neighbourhood is the whole swarm, the best achieved by the whole swarm is called *global best*, $gbest_i$, or sometimes p_g for short.
  - If the neighbourhood is restricted to few particles, the best is called *local best, lbest or* p_l



PSO - Motion

- Each particle adjusts its velocity to move towards its personal best and the swarm neighbourhood best,

- After the velocity is updated, the particle adjusts its position.

PSO - Motion

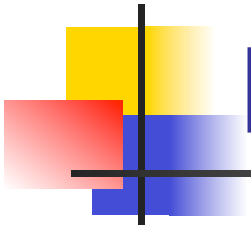
- Equations of motion:

$$v_{t+1}^{id} = w * v_t^{id} + c_1 r_1^{id} \left(pbest_t^{id} - x_t^{id} \right) + c_2 r_2^{id} \left(Nbest_t^{id} - x_t^{id} \right)$$

$$x_{t+1}^{id} = x_t^{id} + v_{t+1}^{id}$$

where

- w is the inertia weight,
- c_1, c_2 are the acceleration coefficients,
- r_1,r_2 are randomly generated numbers in [0, 1],
- t is the iteration number,
- i and d are the particle number and the dimension.



PSO - Motion

- Note that the random numbers are generated for each dimension and not for each particle,

(e.g if the function you are optimizing has 3 variables, the particle will have 3 dimensions)

- If the numbers are generated for each particle, the algorithm is referred to as *linear PSO*, which usually produces sub-optimal solutions in comparison with PSO.

PSO - Motion

$$v_{t+1}^{id} = w * v_t^{id} + c_1 r_1^{id}\left(pbest_t^{id} - x_t^{id}\right) + c_2 r_2^{id}\left(Nbest_t^{id} - x_t^{id}\right)$$

→ Inertia

PSO - Motion

$$v_{t+1}^{id} = w * v_t^{id}$$

$$+ c_1 r_1^{id} \left(pbest_t^{id} - x_t^{id} \right)$$

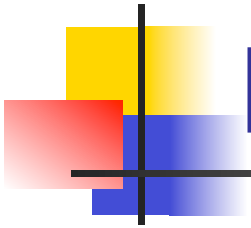
→ Cognitive component

$$+ c_2 r_2^{id} \left(Nbest_t^{id} - x_t^{id} \right)$$

PSO - Motion

$$v_{t+1}^{id} = w * v_t^{id}$$
$$+ c_1 r_1^{id} \left(pbest_t^{id} - x_t^{id} \right)$$
$$+ c_2 r_2^{id} \left(Nbest_t^{id} - x_t^{id} \right)$$

→ Social component

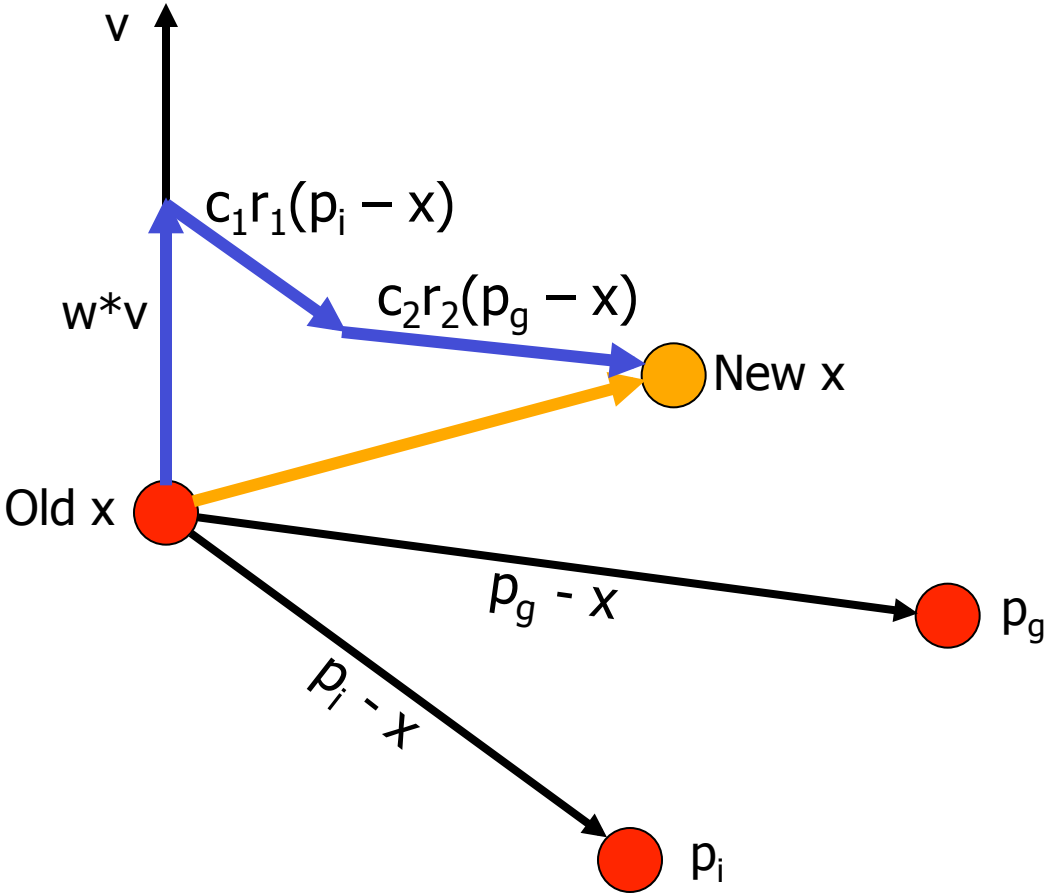


PSO - Motion

- The inertia component accommodates the fact that a bird (particle) cannot suddenly change its direction of movement,

- The c_1 and c_2 factors balance the weights in which each particle:
  - Trusts its own experience, *cognitive component*,
  - Trusts the swarm experience, *social component*.

PSO - Motion



PSO - Motion

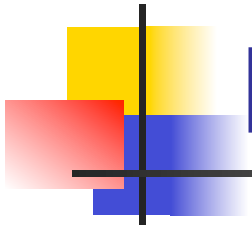
- After that, each particle updates its own personal best (assuming a minimization problem):

$$pbest_{t+1}^{i} = \begin{cases} x_{t+1}^{i} & ,if\ f\left(x_{t+1}^{i}\right) \le f\left(pbest_{t}^{i}\right) \\ pbest_{t}^{i} & ,\text{otherwise} \end{cases}$$

PSO - Motion

- After that, each swarm updates its global best (assuming a minimization problem):

$$\mathrm{Nbest}_{t+1}^{i} = \arg \min_{pbest_{t+1}^{i} \in N} \mathrm{f}\left(\mathrm{pbest}_{\mathrm{t+1}}^{\mathrm{i}}\right)$$



PSO - Motion

- An important factor to set is the maximum velocity allowed for the particles V_{max} :
  - If too high, particles can fly past optimal solutions,
  - If too low, particles can get stuck in local optima.
- Usually set according to the domain of the search space.

PSO – A simple algorithm (Synchronous update)

- Initialize the swarm,
- While *termination criteria* is not met
  - For each particle
    - Update the particle's velocity,
    - Update the particle's position,
    - Update the particle's personal best,

    end for
  - Update the Nbest,

  end while

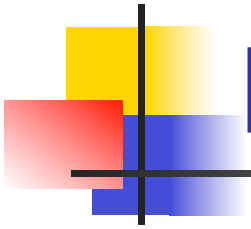
PSO – A different algorithm (Asynchronous update)

- Initialize the swarm,
- While *termination criteria* is not met
  - For each particle
    - Update the particle's velocity,
    - Update the particle's position,
    - Update the particle's personal best,
    - Update the Nbest,

  end for

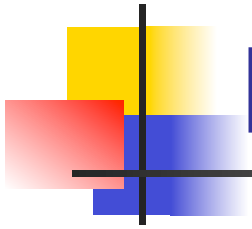
end while

The neighbourhood best update is moved into the particles update loop



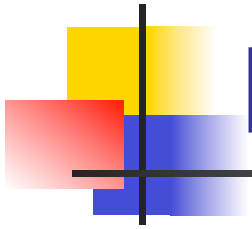
PSO

- *Synchronous* version, if the neighbourhood best is updated after all the population has been updated as well,
- *Asynchronous* version, if the neighbourhood best is updated after every particle,
- Asynchronous version usually produces better results as it causes the particles to use a more up-to-date information.



PSO

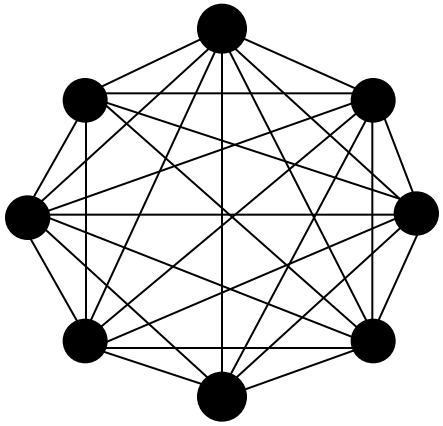
- Termination Criteria can be:
  - Max number of iterations
  - Max number of function evaluations
  - Acceptable solution has been found
  - No improvement over a number of iterations.



PSO - Neighbourhoods

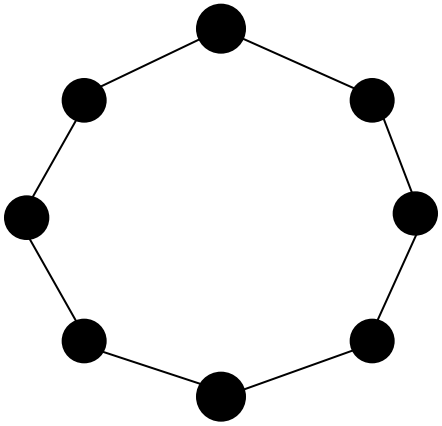
- In PSO, each particle shares its personal best information with other particles,
- Selecting a proper neighbourhood affects the convergence and also helps in avoiding getting stuck at local minima,
- Different neighbourhood topologies have been introduced [9].

PSO - Neighbourhoods



Star topology – global best

best

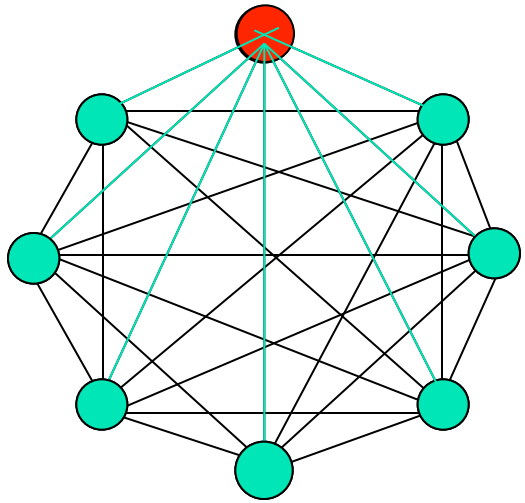


Ring topology – local best

lbest

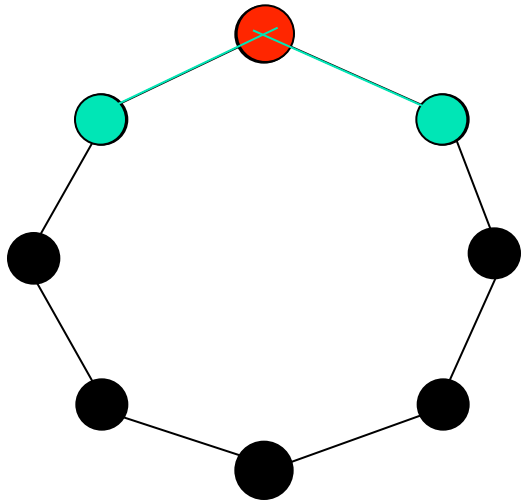
PSO - Neighbourhoods

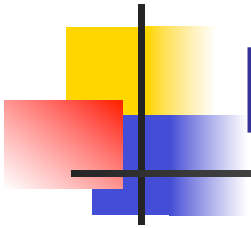
- *gbest model:* each particle is influenced by all the other particles,
- The fastest Propagation of information in a population,
- Particles can get stuck easily in local minima.



PSO - Neighbourhoods

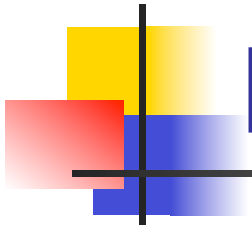
- *lbest model:* each particle is influenced only by particles in its own neighbourhood,
- The propagation of information is the slowest,
- Doesn't get stuck easily in local minima but might increase the computational cost.





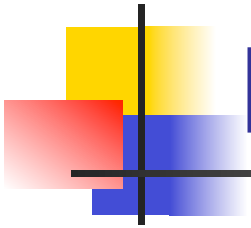
PSO - Neighbourhoods

- The most obvious way to select the neighbours for a certain particle *m* is to choose the particles that are closest to it in the search space (physical neighbors)
- The notion of closest is based on the distance in the Cartesian space,
- This approach might be computationally expensive as distances has to be computed each time the particle changes its position.



PSO - Neighbourhoods

- If the particles are kept in a matrix data structure for example,
- The most commonly used approach is to pick the particles that are stored next to *m* in the matrix (social neighbors)
- The performance is affected by the size of the neighbourhood selected (2, 4, 6, …).

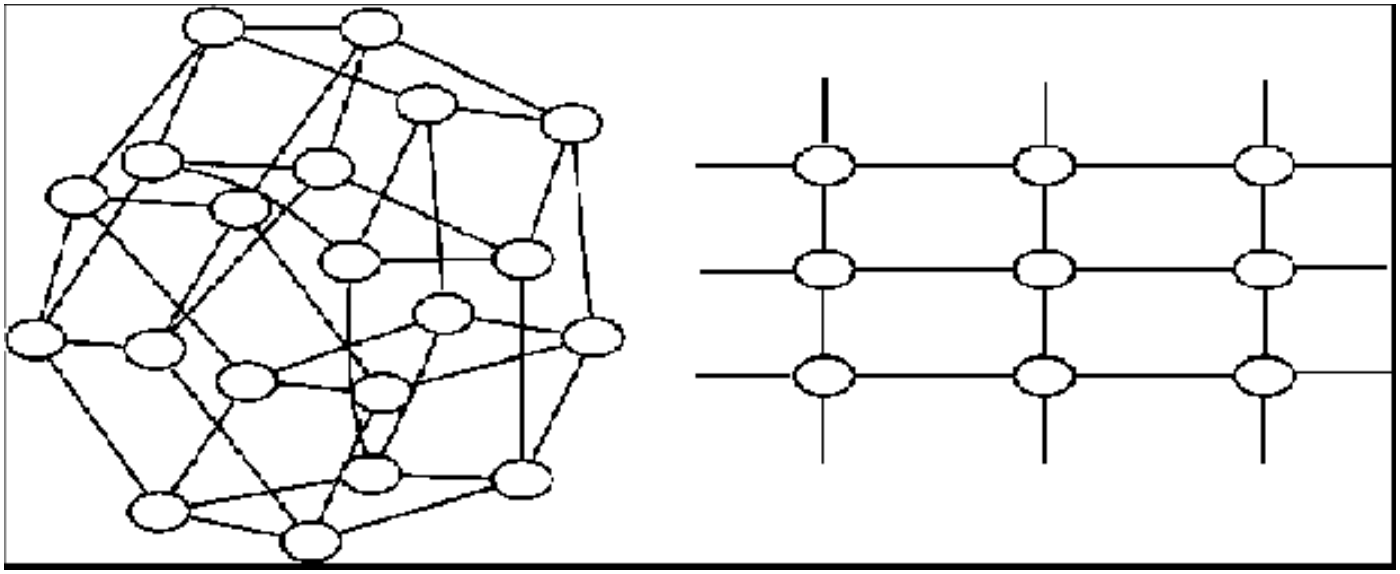


PSO - Neighbourhoods

- The most successful neighbourhood structure was the square topology (Von Neumann model),

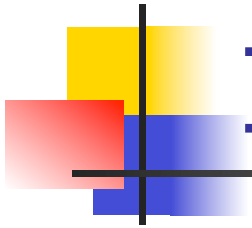
- Formed by arranging the particles in a grid and connecting the neighbours above , below and to the right and left.

PSO - Neighbourhoods



The complete population A local region

The Von Neumann model [10]



Initialization

- For each particle need to initialize the particle position and velocity.
- Particles positions can be initialized randomly in the range.
- Particles velocities can be initialized to zero or small values, large values will result in large updates which may lead to divergence
- Personal best position is initialized to the particle’s initial position

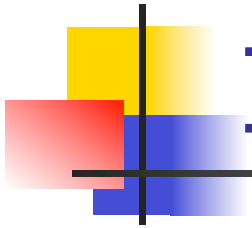
Initialization

- Parameters w, c_1, c_2, r_1, r_2

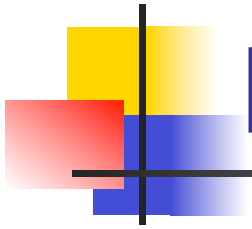
$$v_{t+1}^{id} = w * v_t^{id} + c_1 r_1^{id} \left(pbest_t^{id} - x_t^{id} \right) + c_2 r_2^{id} \left(Nbest_t^{id} - x_t^{id} \right)$$

- Using $c_1 = 0$ reduces the velocity model to Social-only model (particles are all attracted to *Nbest*)
- Using $c_2 = 0$ reduces to cognition-only model (particles are independent hill climbers)

Initialization



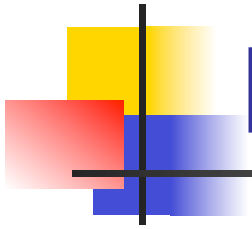
- In most applications $c_1 = c_2$
- Small values will result in smooth trajectories
- High values cause more acceleration with abrupt movement towards or past good regions
- The value of w is important to balance exploration and exploitation
- Large values promote exploration and small promote exploitation (allowing more control to cognitive and social components)



PSO - Convergence

- An important question is: What are the suitable parameter values that guarantee the swarm convergence?

- The wrong settings can cause the particles to explode out of the search space,



PSO - Convergence

- One of the simplest theoretical studies for PSO was proposed by Trelea in 2003 [7],
- The study followed the same steps taken in dynamic systems theory,
- It was carried using a deterministic PSO algorithm, randomness was removed.

PSO - Convergence

- The random numbers were replaced by their expected values, so for a one-dimensional one-particle system:

$$r_1 = r_2 = \frac{1}{2}$$

$$v_{t+1} = w * v_t + \frac{c_1}{2}\left(pbest_t - x_t\right) + \frac{c_2}{2}\left(gbest_t - x_t\right)$$

$$x_{t+1} = x_t + v_{t+1}$$

PSO - Convergence

- Let:

$$c = \frac{c_1 + c_2}{2}$$

$$p = \frac{c_1}{c_1 + c_2} pbest + \frac{c_2}{c_1 + c_2} gbest$$

PSO - Convergence

- Hence, the equations of motion would be:

$$v_{t+1} = w * v_t + c(p - x_t)$$

$$x_{t+1} = x_t + v_{t+1}$$

PSO - Convergence

- The equations could be rewritten in matrix form:

$$y_{t+1} = Ay_t + Bp$$

where

$$y_t = \begin{bmatrix} x_t \\ v_t \end{bmatrix}, A = \begin{bmatrix} 1-c & w \\ -c & w \end{bmatrix}, B = \begin{bmatrix} c \\ c \end{bmatrix}$$

PSO - Convergence

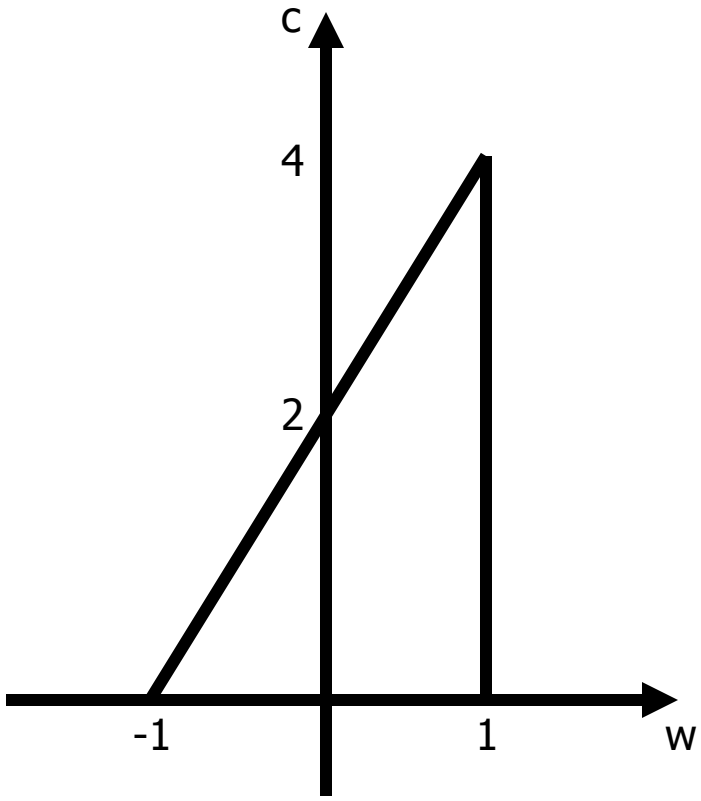
- By analyzing the characteristic equation, we can find the conditions necessary for stability:

$$w < 1,$$

$$c > 0,$$

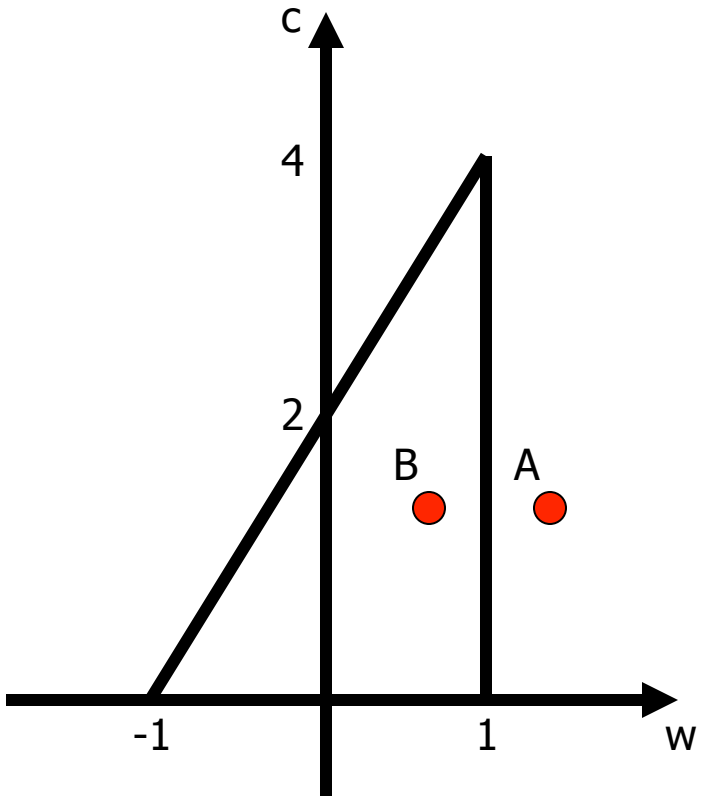
$$2 * w - c + 2 > 0$$

- The convergence domain would be inside the triangle shown.

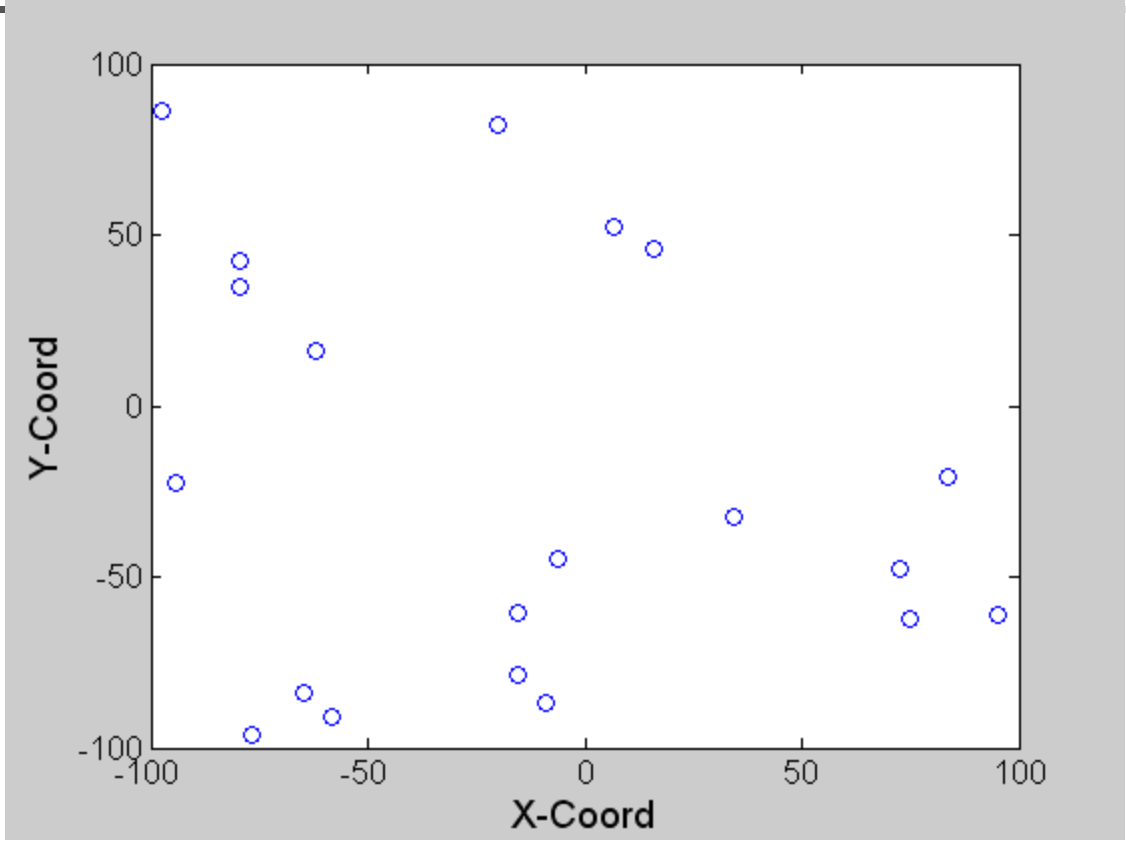


PSO - Convergence

- Two simulations are run to optimize the Spherical function using 20 particles in a domain range of [-100, 100],
- The values used for [*c, w*] are:
  - A=[1.2, 1.49] for the first run,
  - B=[0.79, 1.49] for the second run.

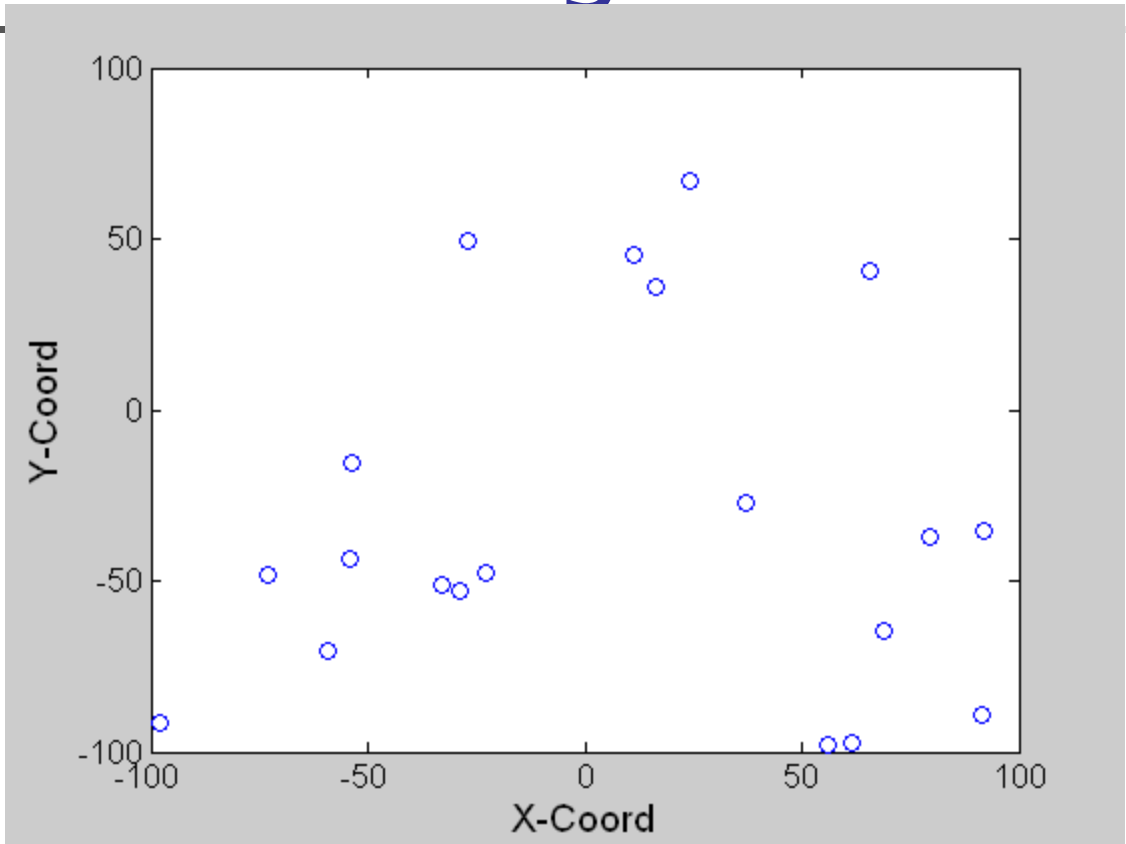


PSO - Convergence



20 particles Using parameters A

PSO - Convergence



20 particles Using parameters B

PSO - Convergence

- The widely used set of parameters is proposed by Clerc and Kennedy in 2003 [8]:

$$w = 0.792,$$

$$c1 = c2 = 1.4944$$

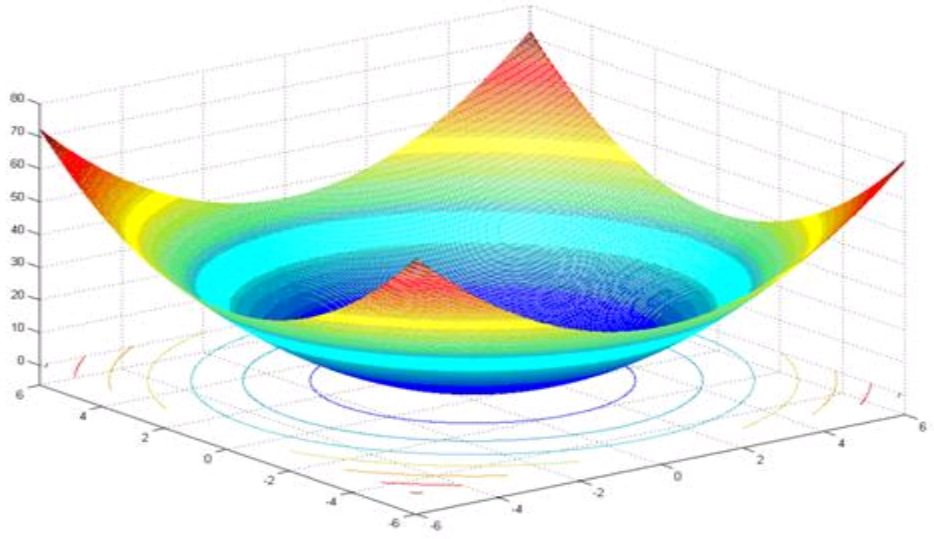
- The standard PSO version could be downloaded from:

  http://www.particleswarm.info/standard_pso_2007.c

PSO – A Comparison with GAs

- Spherical:

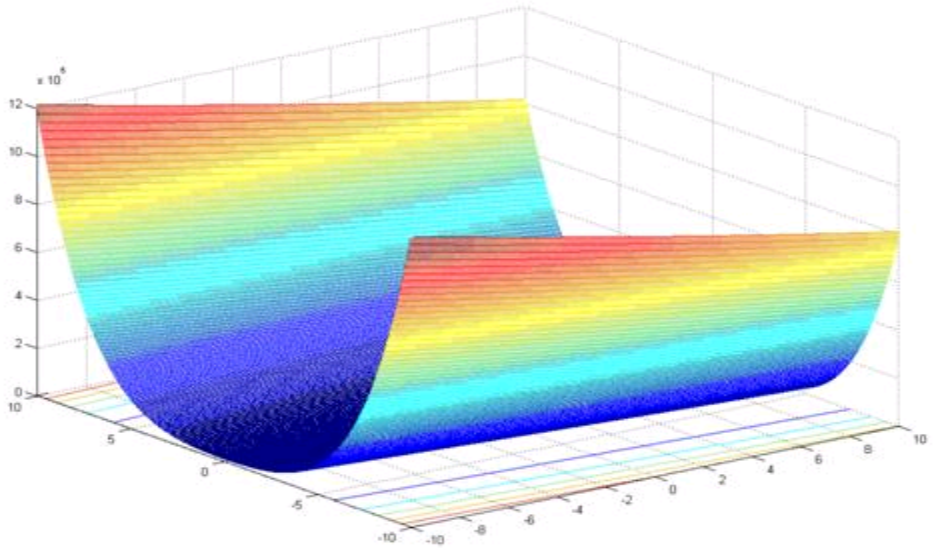
$$f(x) = \sum_{i=1}^{d} x_i^2$$



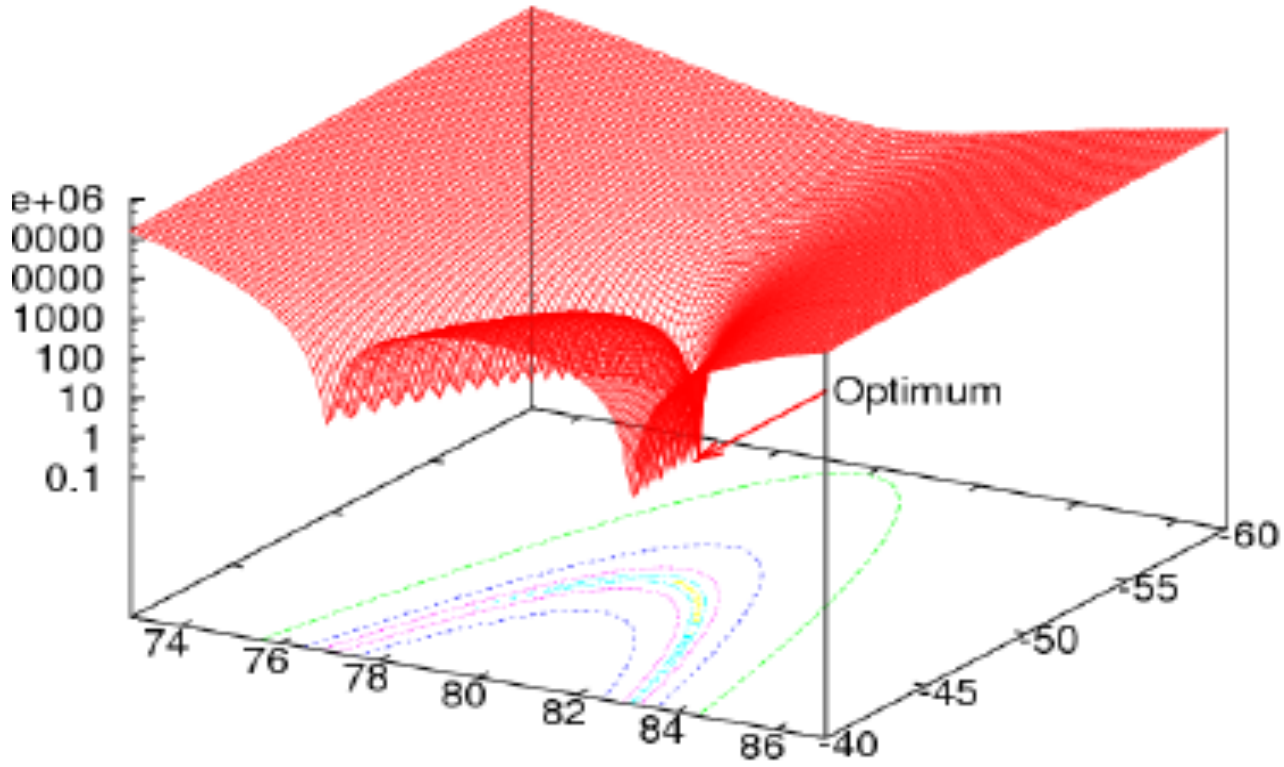
PSO – A Comparison with GAs

- Rosenbrock:

$$f(x) = \sum_{i=1}^{d-1} \left[(1 - x_i)^2 + 100 \left(x_{i+1} - x_i^2 \right)^2 \right]$$



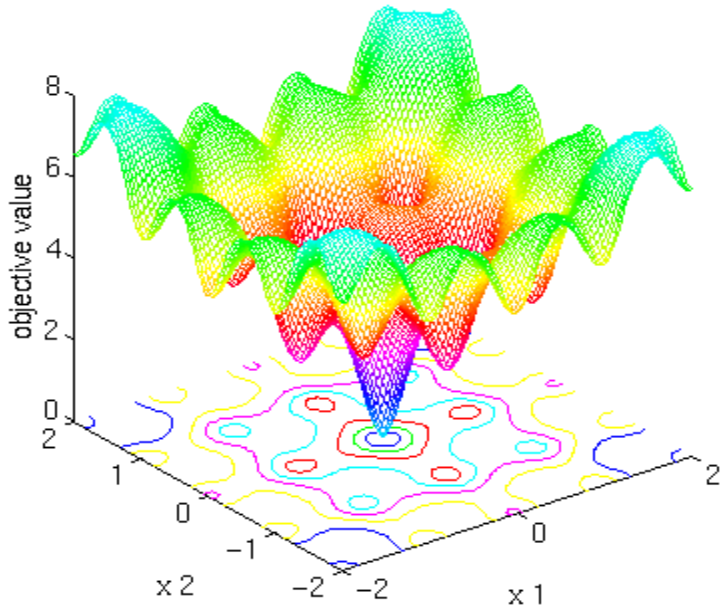
Rosenbrock


e+06
0000
0000
1000
100
10
1
0.1
Optimum
74
76
78
80
82
84
86
-40
-45
-50
-55
-60

PSO – A Comparison with GAs

- Ackley:

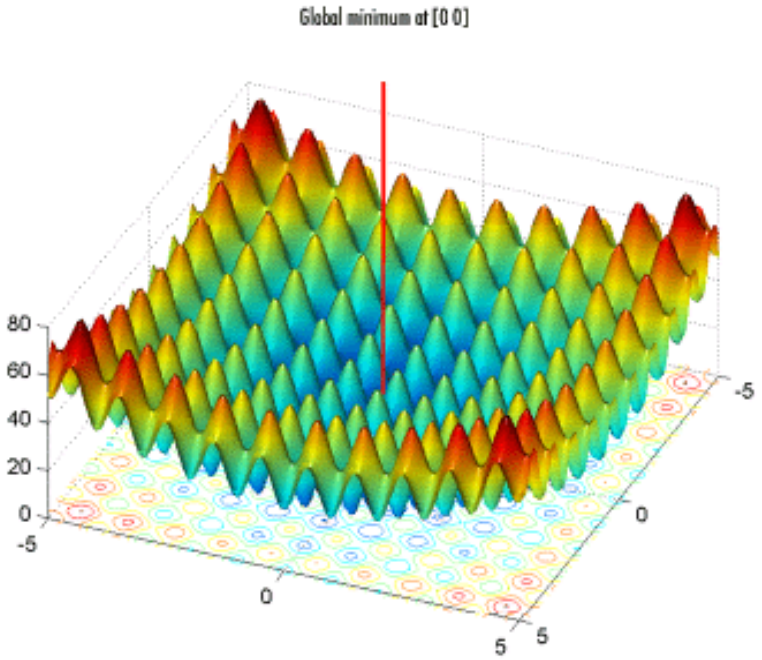
$$f(x) = -20\exp\left(-0.2\sqrt{\frac{1}{d}\sum_{i=1}^{d} x_i^2}\right) - \exp\left(\frac{1}{n}\sum_{i=1}^{d}\cos(2\pi x_i)\right) + 20 + e$$

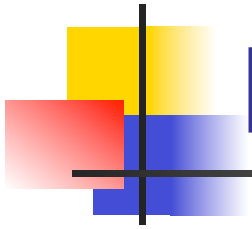


PSO – A Comparison with GAs

- Rastrigin:

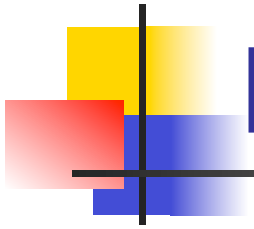
$$f(x) = \sum_{i=1}^{d} x_i^2 - 10\cos(2\pi x_i) + 10$$





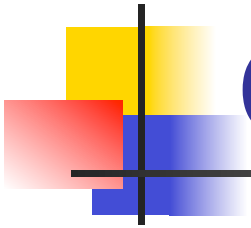
PSO - A comparison with GAs

- A comparison is made between PSO and GAs using the four functions (Spherical, Rosenbrock, Ackley and Rastrigin):
  - Both algorithms use 10 particles (individuals) and run for 1000 iterations, using Clerc and Kennedy parameters.
  - For a dimensionality of 10,
  - The results are the averages reported over 20 runs.



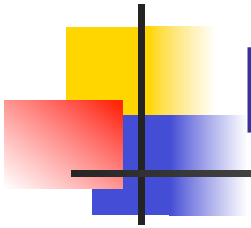
PSO – A Comparison with GAs

Benchmark	GA - Elitism	PSO - *gbest*
Spherical	0.0099	**2.3273e-17**
Rosenbrock	**5.5760**	9.6467
Ackley	0.9451	**0.3047**
Rastrigin	38.2186	**18.7730**



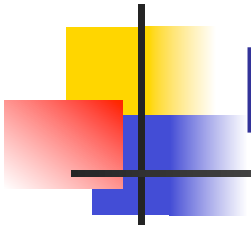
Outline

- Introduction,
- PSO,
- Discrete PSO
  - Binary PSO,
  - Permutation PSO,
- Applications,
- Cooperation,
- References.



Discrete PSO

- PSO was originally developed for continuous-valued spaces
- Many problems are defined for discrete valued spaces
- Examples: Feature selection, TSP, Assignment problems, Scheduling,..
- Changes to PSO can be as simple as discretization of the position vectors or complex as redefining of the arithmetic operations (addition, multiplication)



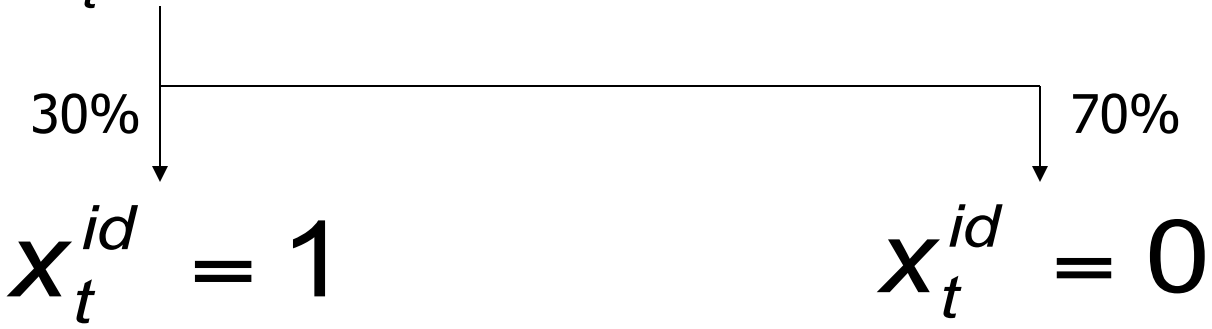
Binary PSO

- A binary PSO version was proposed in [11],
- Each particle represents a position in the binary space,
- Each element can take the value of 0 or 1.

Binary PSO

- Velocities are defined as probabilities that one element will be in one state or the other,

- If $V_t^{id} = 0.3$



Binary PSO

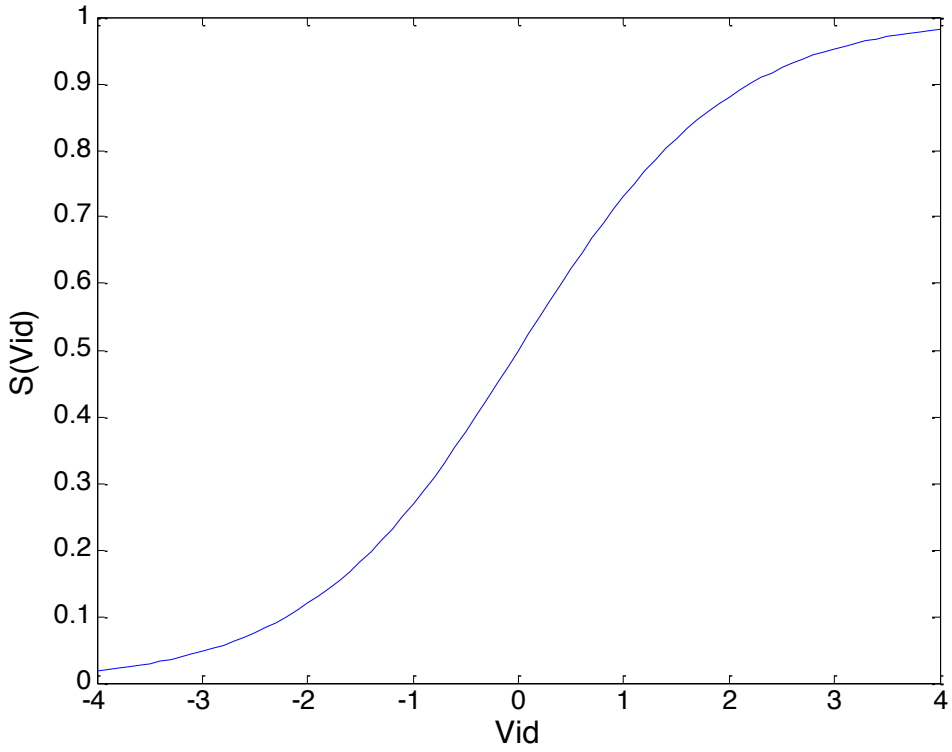
- Since velocities represent probabilities, the values of the velocity elements need to be restricted in the range [0,1],

- This is done by using the sigmoid function:

$$sig\left(V_t^{id}\right) = \frac{1}{1 + e^{-V_t^{id}}}$$

Binary PSO

- The sigmoid function:



Binary PSO

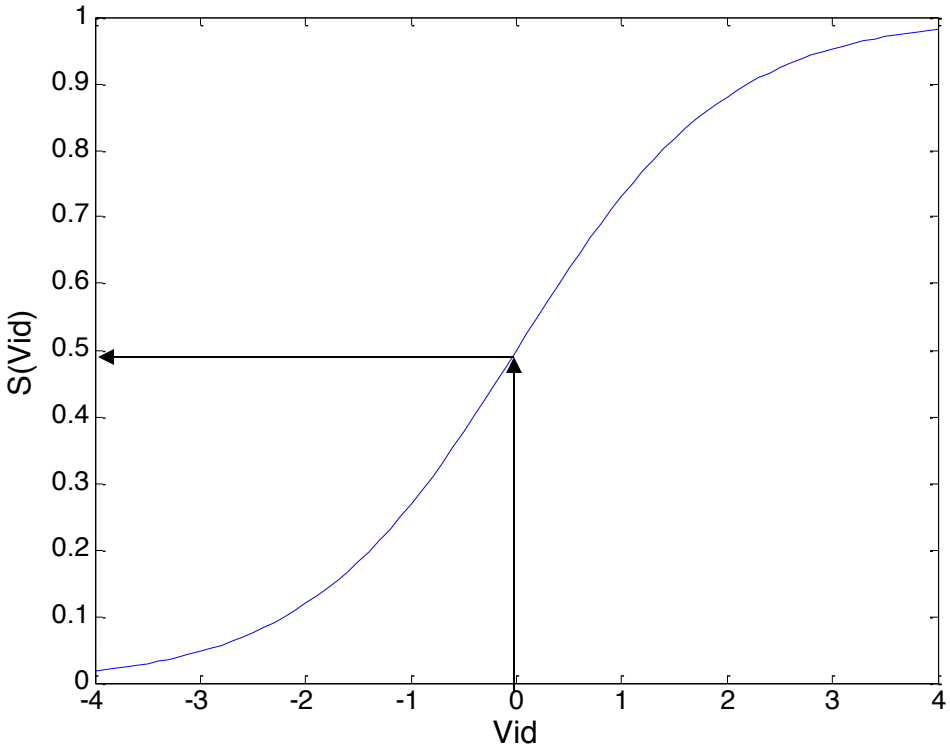
- The position update equation then becomes:

$$x_{t+1}^{id} = \begin{cases} 1 & ,if\ \mathrm{r} < sig\left(v_{t+1}^{id}\right) \\ 0 & ,\text{otherwise} \end{cases}$$

where r is a randomly generated number in [0, 1].

Binary PSO

- The sigmoid function:

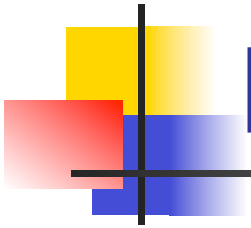


Binary PSO

- This means that:

$$prob\left(x_{t+1}^{id}=1\right)=\begin{cases}0.5 & ,v_{t+1}^{id}=0\\ <0.5 & ,\ v_{t+1}^{id}<0\\ >0.5 & ,\ v_{t+1}^{id}>0\end{cases}$$

- Note that the velocity components could remain as real-valued numbers using the original equation, but fed to the sigmoid function before updating the position vector.



Binary PSO

- In [11], a comparison was made between the binary PSO and three version of GAs:
  - Crossover only GA (GA_c),
  - Mutation only GA (GA_m),
  - A regular GA.
- The objective was to optimize different functions that were generated using a random function generator.

Binary PSO

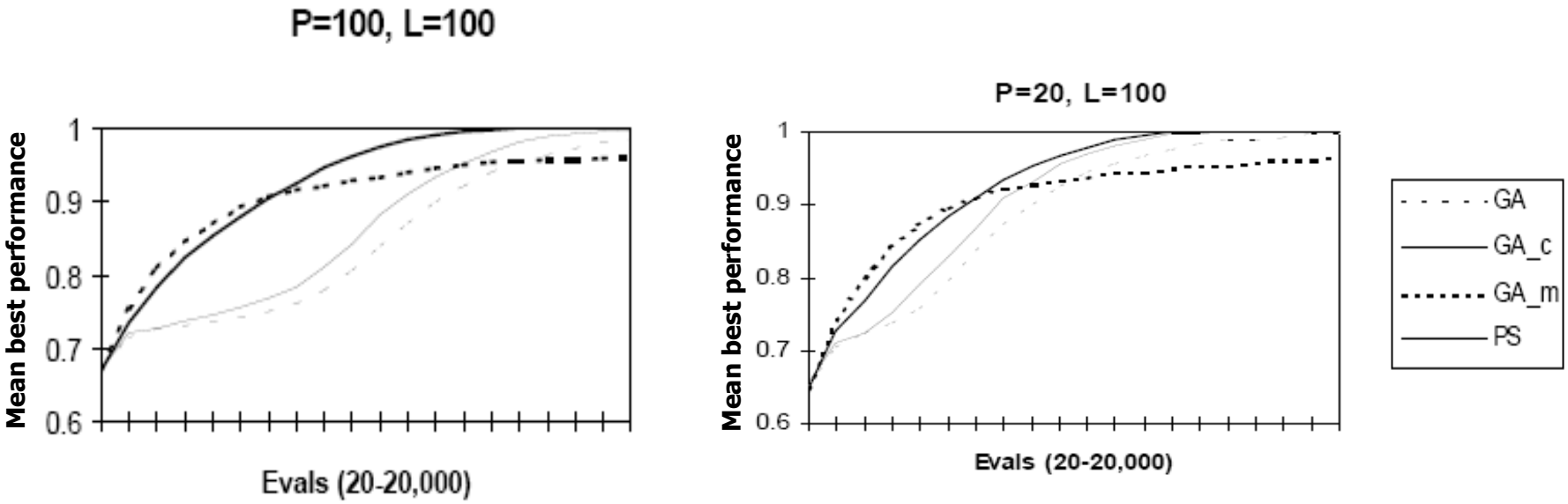
- This allows the creation of random binary problems with some specified characteristics:
  - Number of local optima, bit strings
  - Dimension, no of components (bits)
  - Fitness function

$$f(c) = \frac{1}{L} \max_{i=1}^{P} \{L - Hamming(c, Peak_i)\}$$

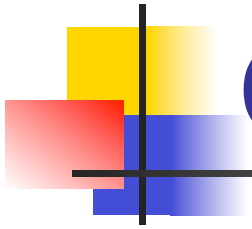
- In that study, the binary PSO was the only algorithm that found the global optimum on every single trial, regardless of problem features.

Binary PSO

- The experiments show that the binary PSO progressed faster than the other algorithms.

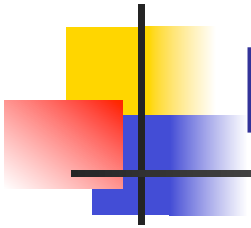


P is the number of peaks and L is the binary vector length



Outline

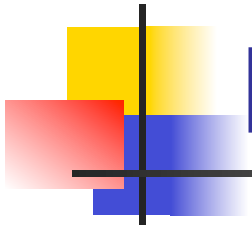
- Introduction,
- PSO,
- Discrete PSO
  - Binary PSO,
  - Permutation PSO,
- Applications,
- Cooperation,
- References.



Permutation PSO

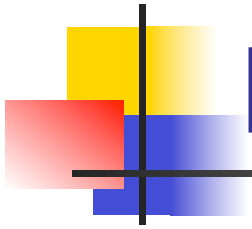
- Several attempts have been made to apply PSO to permutation problems,

- The difficulty of extending PSO to such problems is that the notions of velocity and direction have no natural extensions for these problems.



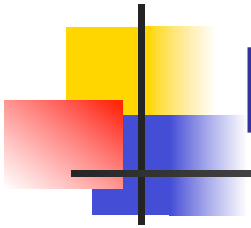
Permutation PSO

- In [12], PSO was applied to solve the TSP,
- The position of a particle was the solution to a problem (permutation of cities),
- The velocity of a particle was defined as the set of swaps to be performed on a particle.



Permutation PSO

- Three operations were re-defined for the new search space:
  - Adding a velocity to a position,
  - Subtracting two positions,
  - Multiplying a velocity by a constant.



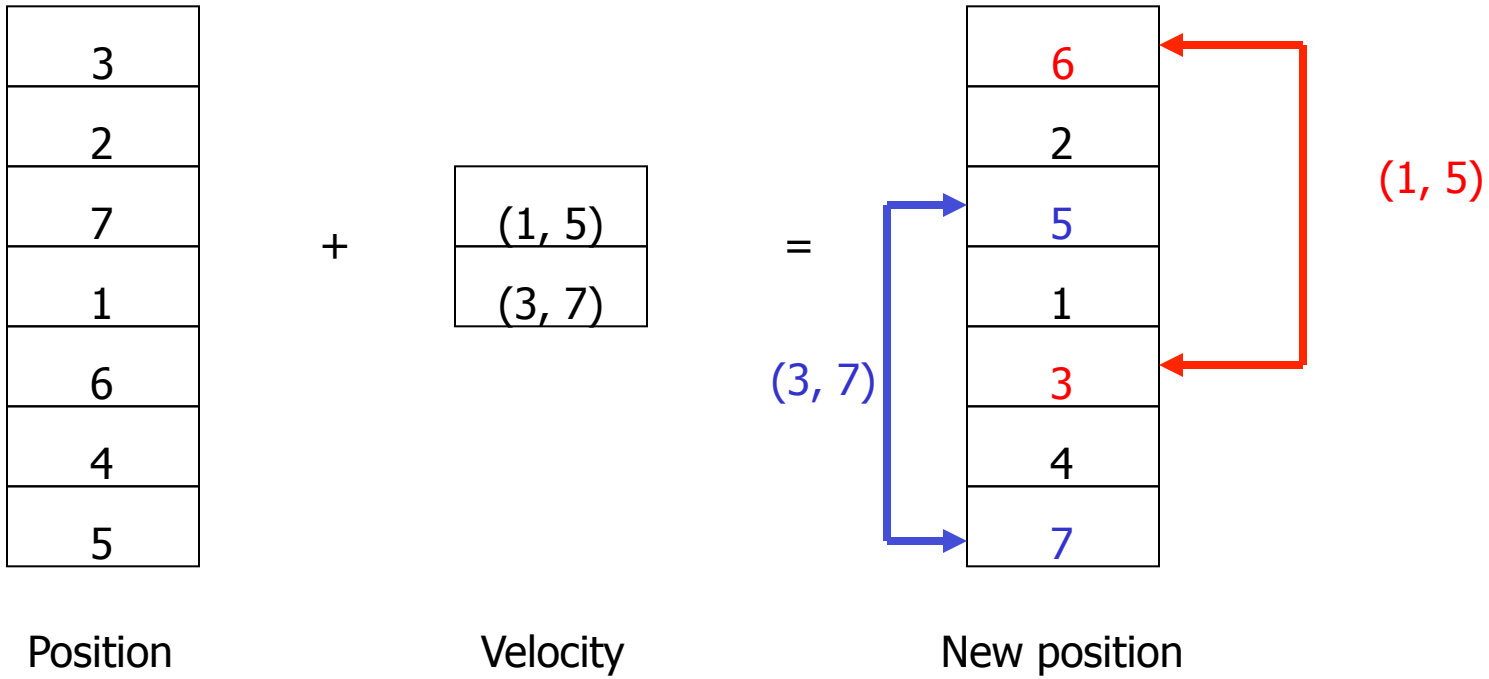
Permutation PSO

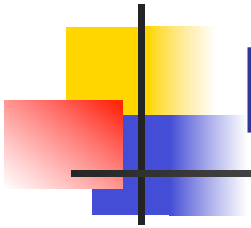
- Adding a velocity to a position:

  the operation is performed by applying the sequence of swaps defined by the velocity to the position vector.

Permutation PSO

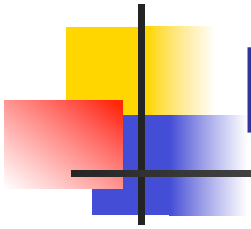
- Adding a velocity to a position:





Permutation PSO

- Subtracting two positions:
  - Subtracting two positions should produce a velocity,
  - This operation produces the sequence of swaps that could transform one position to the other.



Permutation PSO

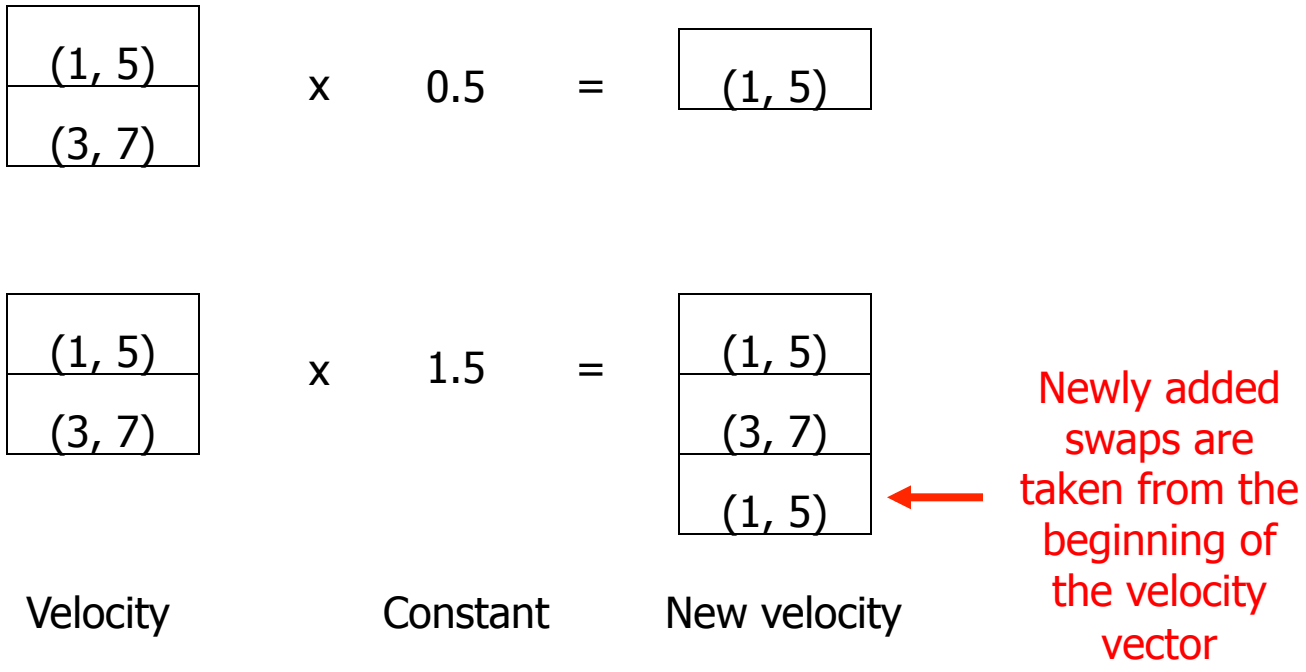
- Multiplying a velocity by a constant:

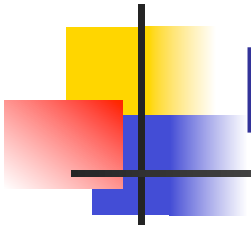
  This operation is performed by changing the length of the velocity vector (number of swaps) according to the constant c :

  - If $c = 0$, the length is set to zero,
  - If $c < 1$, the velocity is truncated,
  - If $c > 1$, the velocity is augmented.

Permutation PSO

- Multiplying a velocity by a constant:





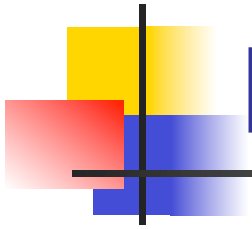
Permutation PSO

- The permutation PSO was successfully applied to TSP,
- Using 16 particles and a neighbourhood of 4 particles, it finds the optimal solution for problem with size of 17 (br17.tsp) using 7990 tour evaluations.

Permutation PSO

- Using a social neighbourhood (close particles in the swarm matrix) is less expensive than using physical neighbourhood (close in the search space).

Swarm size	Neighbourhood size	Neighbourhood type	Logical/arithmetic operations
16	4	Social	4.8M
16	4	Physical	6.3M

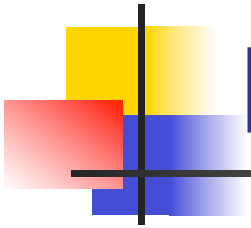


Permutation PSO

- However, applying it to larger problems might be computationally expensive (due to the handling of multiple solutions).

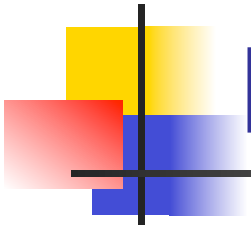
Permutation PSO

- A different PSO approach was applied to the TSP in [15],
- A continuous PSO version was used,
- In order to evaluate the particle fitness, they performed a *space transformation* from a particle in the continuous domain to a permutation in the solution space.



Permutation PSO

- The used rule known as *Great Value Priority (GVP)*,
- First, all the elements in the position vector (along with their indices) were sorted to get a sorted list in descending order,
- The sorted indices were taken as the permutation.



Permutation PSO

- The idea was that if x_i had the highest value in the position vector, it means that city number *i* comes first in the permutation vector,

- This transformation was carried before calculating the particles fitness.

Permutation PSO

- Example:

$$if \quad x = [0.2, 0.3, 0.1, 0.8]$$

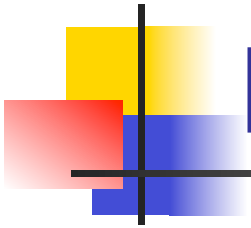
then the sorted list would be:

$$\begin{matrix} 4 & 2 & 1 & 3 \end{matrix}$$

$$x_{sorted} = [0.8, 0.3, 0.2, 0.1]$$

and the permutation would be:

Tour to be evaluated and its cost is the particles' x fitness → $P = [4, 2, 1, 3]$



Permutation PSO

- The work also experimented with applying local search to the permutation with a certain probability to get a better one,
- The local search was applied by selecting two cities to swap,
- If a better tour is found, in order to go back to the continuous domain, the same swap is applied to the original particle x.

Permutation PSO

- Example:

  if $P = [4, 2, 1, 3]$ and $X = [0.2,\ 0.3,\ 0.1,\ 0.8]$

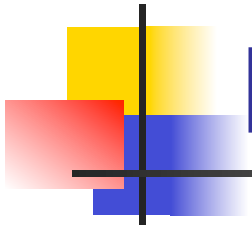
  and the local search produced the better tour:

$$P = [2, 4, 1, 3]$$

  then the new position would be:

$$x = [0.2, 0.8, 0.1, 0.3]$$

  by swapping the same elements

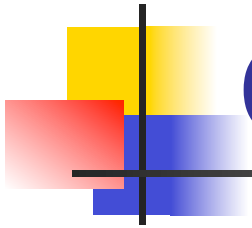


Permutation PSO

- The PSO and PSO_LS both were applied to 4 TSP problems using:
  - 50 particles and 2000 iterations,
  - The position constrained in [-1, 1],
  - The velocity constrained in [-0.1, 0.1],
  - w=1, c_1=c_2=2,
  - A local search probability of 1%,
  - The results are the averages taken over 10 runs.

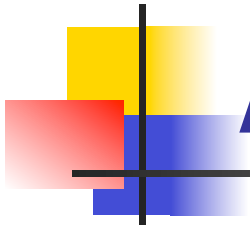
Permutation PSO

Instance	PSO	PSO_LS	Optimal Solution
burma14	33.6948	30.8785	30.8785
eil51	582.501	459.273	426
berlin52	8100.105	7938.041	7542
eil76	588.712	564.523	538



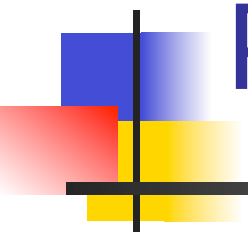
Outline

- Introduction,
- PSO,
- Discrete PSO,
  - Binary PSO
  - Permutation PSO,
- Applications,
- Cooperation,
- References.



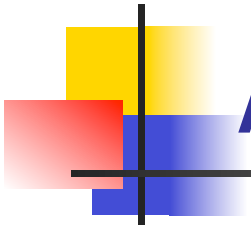
Applications

- PSO has been successfully applied to many applications including:
  - FPGA Placement
  - Neural network training,
  - Robot motion planning.
  - Clustering.



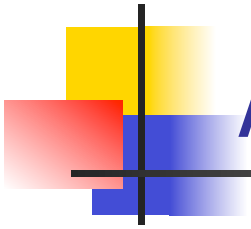
PSO

Examples



Applications

- PSO has been successfully applied to many applications including:
  - FPGA Placement
  - Neural network training,
  - Robot motion planning.
  - Clustering.

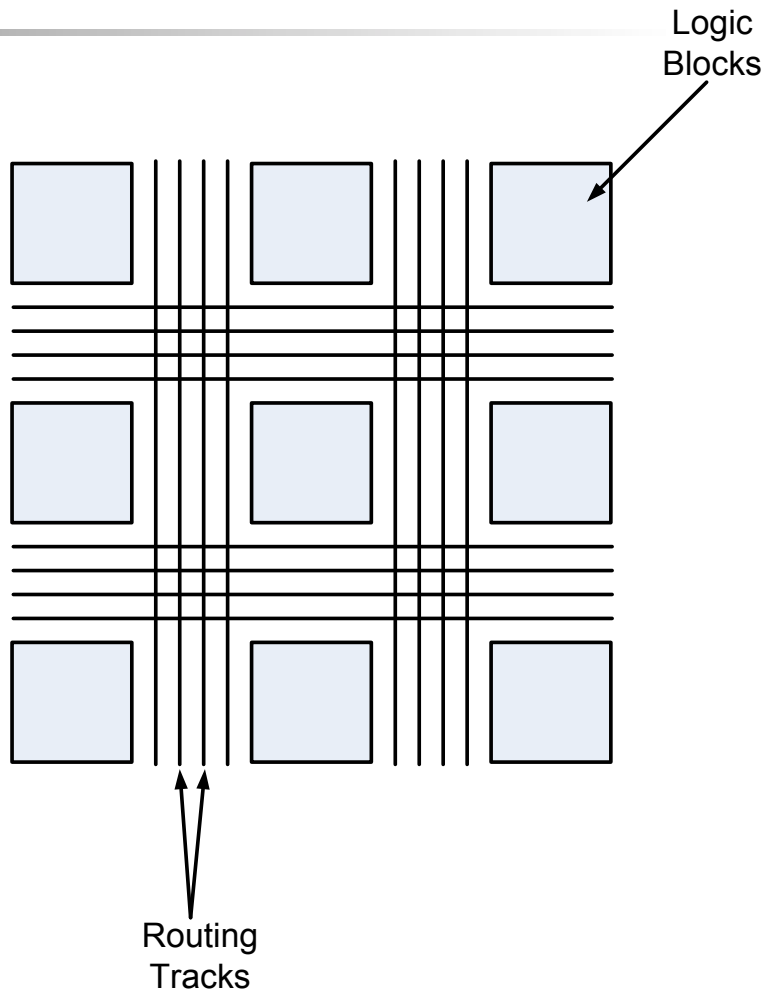


Application 1- FPGA Placement

- PSO was applied to the FPGA placement problem and compared to Versatile Place and Route (VPR),
- VPR is an academic tool adopting a simulated annealing approach,
- The objective was to minimize the total wire length.

FPGA Placement

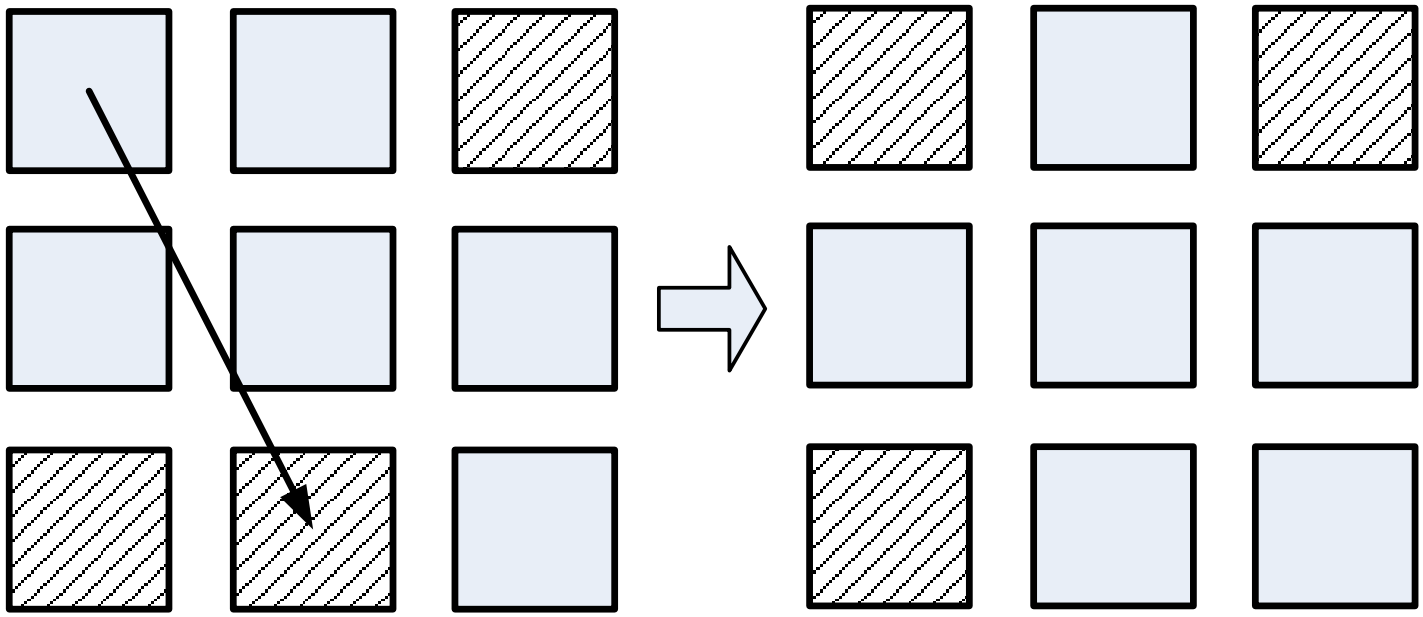
- FPGA placement is performed in discrete locations,
- All logic blocks have the same height and width.



From [1]

FPGA Placement Perturbations

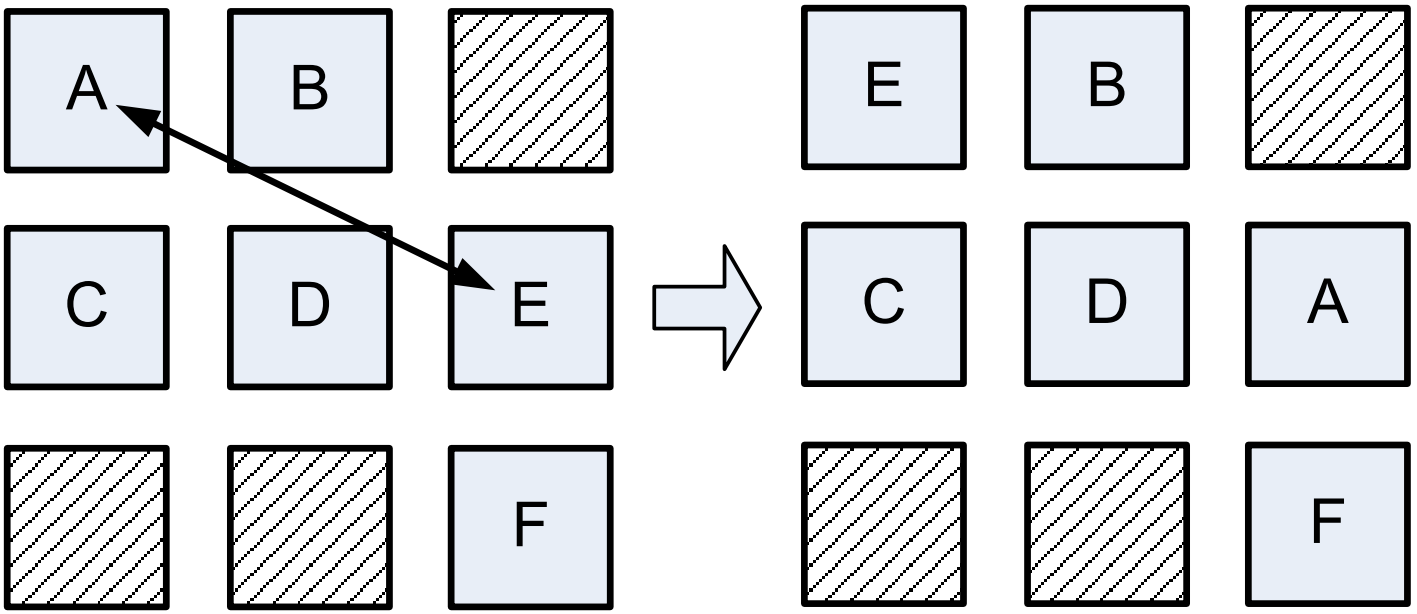
- Move to an empty location:



From [2]

FPGA Placement Perturbations

- Swap with another block:

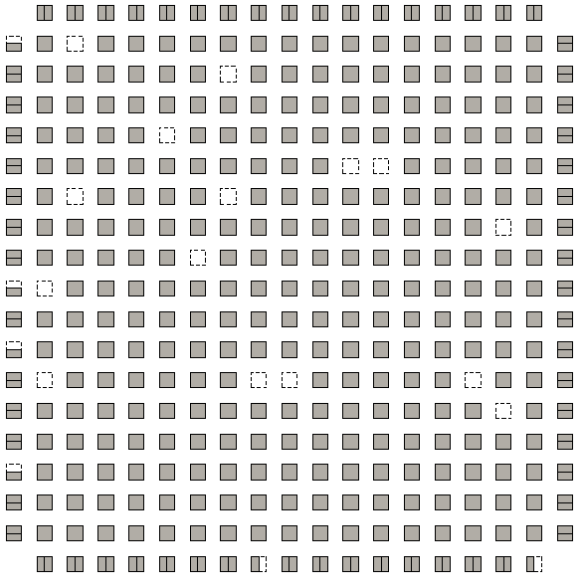


From [1]

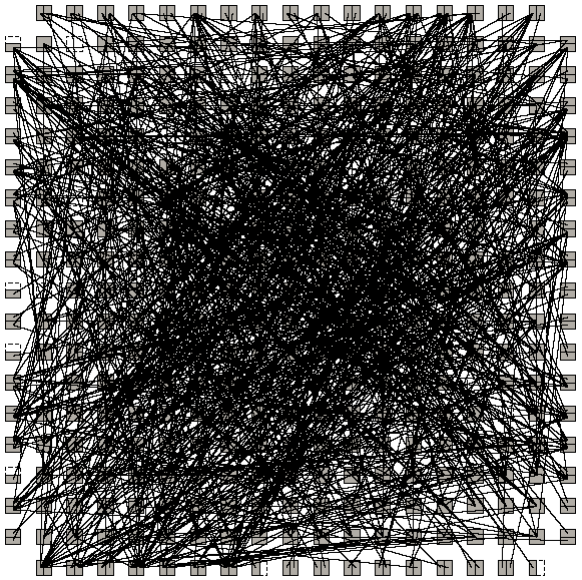
FPGA Placement

- Placement by Versatile Place and Route (VPR) tool:
  - VPR uses SA,
  - VPR is the core of Altera's placement tools

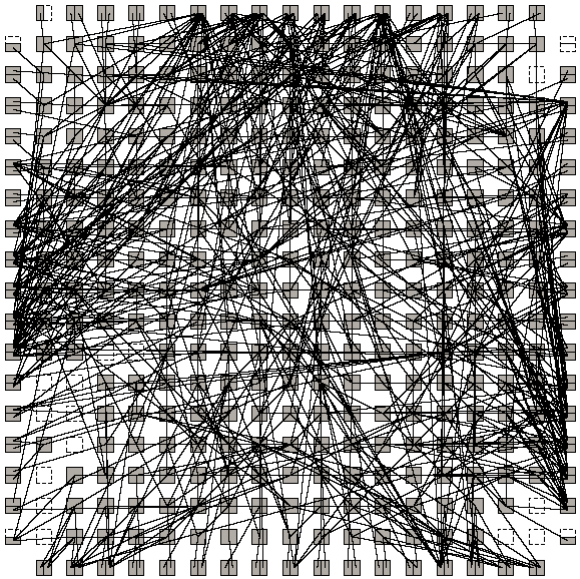
Initial placement: wire-length=74.69



Initial placement: wire-length=74.69



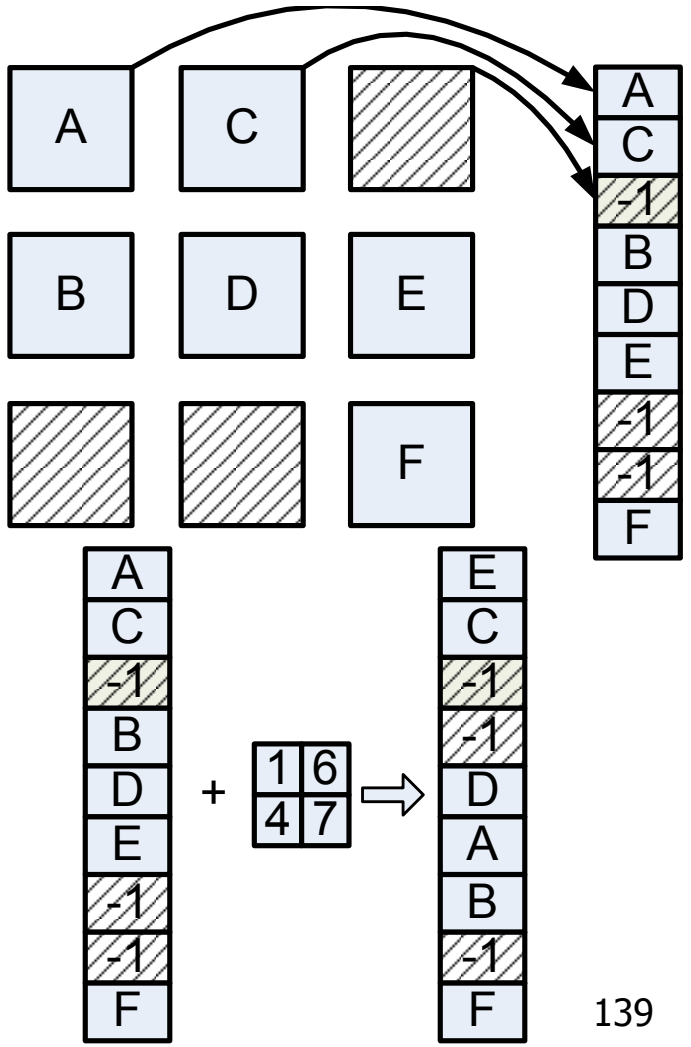
Final placement: wire-length=30.56



From [2]

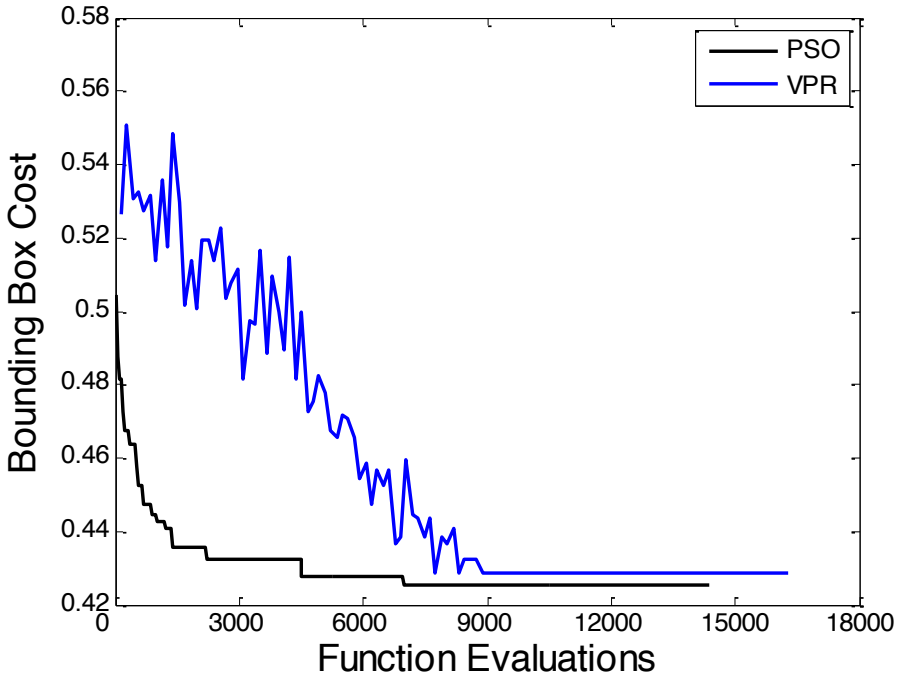
FPGA Placement Using PSO

- Each particle contains information of all the available locations,
- Velocities are swap pairs.

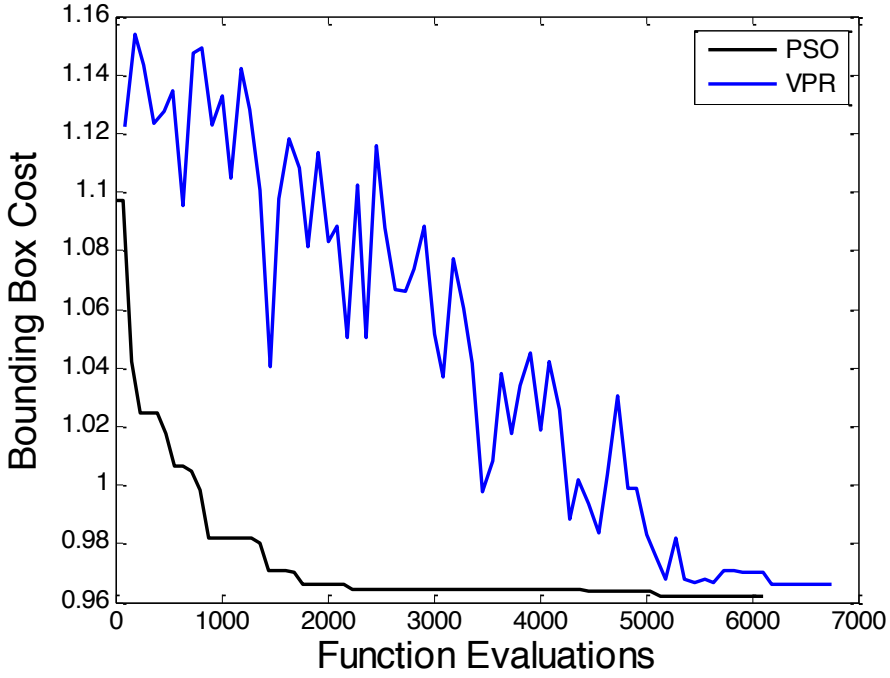


FPGA Placement Using PSO

- PSO has faster convergence than SA.



cm42a: problem size=36

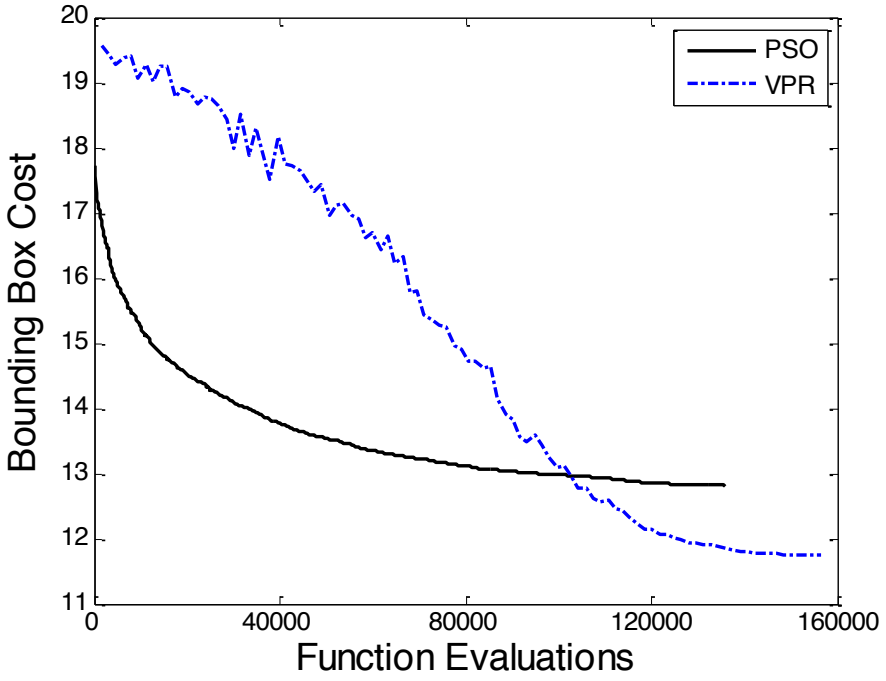


dk27: problem size=80

From [1]

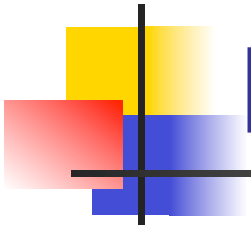
FPGA Placement Using PSO

- For problems with larger dimensionality, PSO gets stuck in local minima,
- PSO still has faster convergence rate than SA.



s832: problem size=225

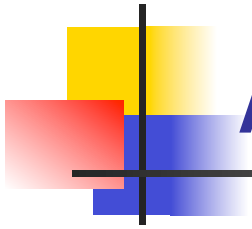
From [1]



FPGA using PSO

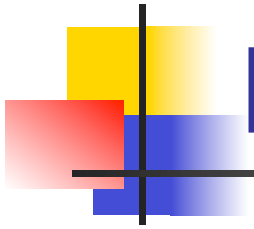
- PSO always starts from a better solution due to the initialization of a number of particles,

- However, it has a higher computational cost than SA due to the handling of a complete swarm.



Application 2 - NNT

- NN performs a mapping from a set of input data to one or more output nodes,
- It's a structured set of nodes that are fully or partially connected,
- Each node is a processing unit that gets stimulated by:
  - An external input,
  - Another node.



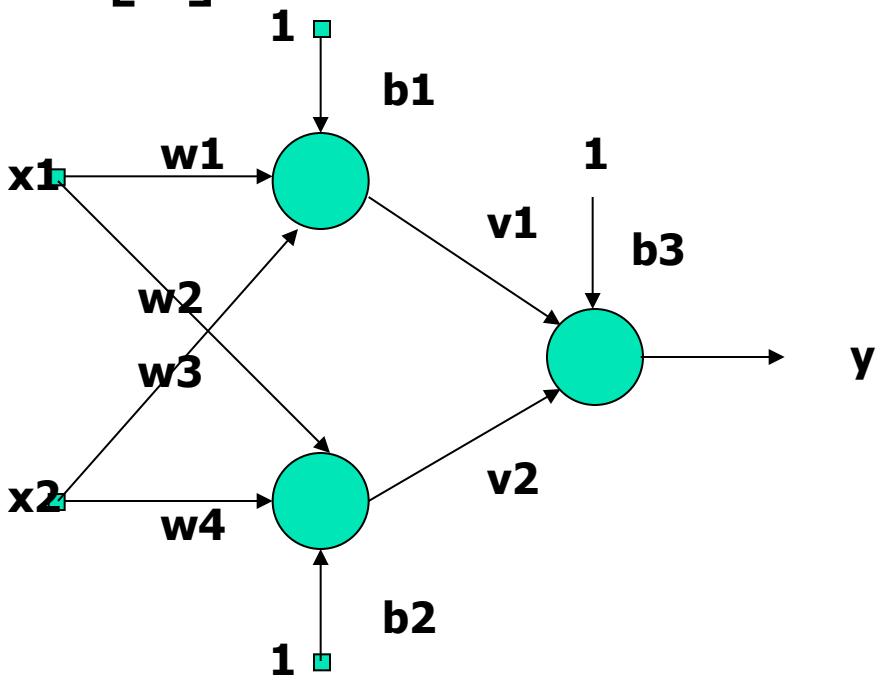
NNT

- The node produces an output signal that reaches:
  - An output,
  - Another node.

- It's required to find the appropriate set of weights that will lead to a satisfactory network behaviour.

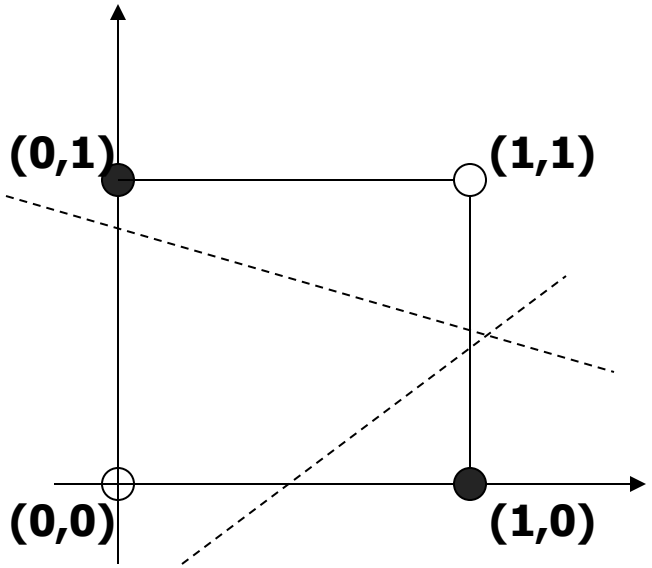
NNT

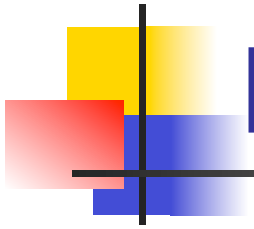
- PSO proposed for optimizing the weights of an ANN performing the XOR operation in [3].



NNT

x1	x2	y
0	0	0
0	1	1
1	0	1
1	1	0





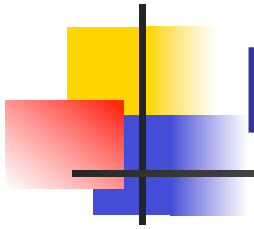
NNT

- How is the problem formulated?
  - PSO could be used to optimize the weights, each particle will be a 9-dimensional vector:

$$x = \left[w_1, w_2, w_3, w_4, b_1, b_2, b_3, v_1, v_2 \right]$$

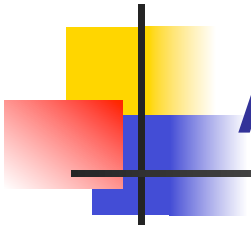
  - The velocity is a 9-dimensional vector of continuous values,
  - The objective is to minimize the error between the desired and the actual outputs of the network.

- It starts by randomly initializing a group of particles and let them move in the search space until an acceptable error is reached.



NNT

- PSO was effective as the back-propagation algorithm in training this NN,
- A swarm of 20 particles was able to train network to an error of < 0.05 in an average of 30.7 iterations.

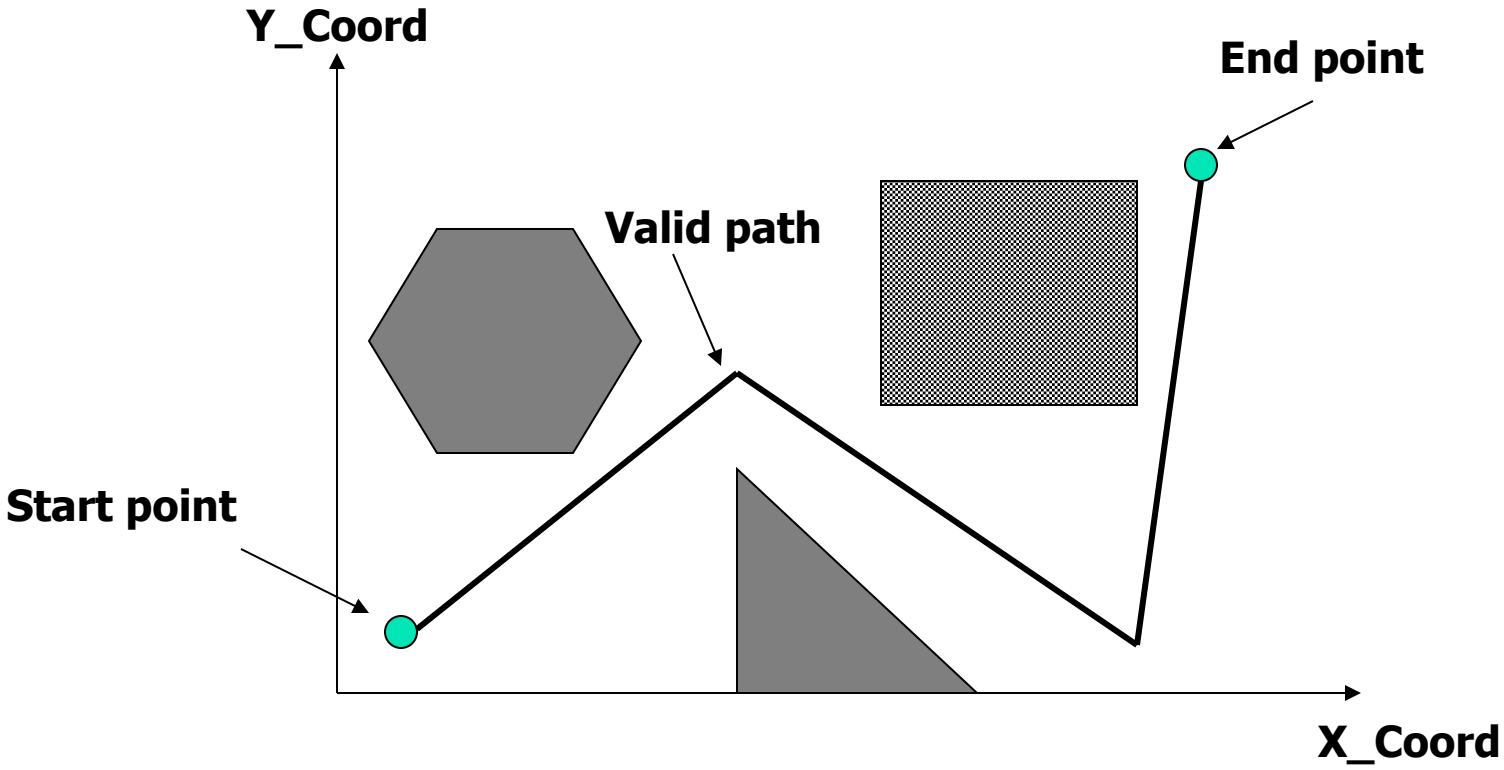


Application 3 - RMP

- Robot motion planning involves finding the path a robot needs to traverse from a given start point to a given end point while avoiding obstacles along the way,

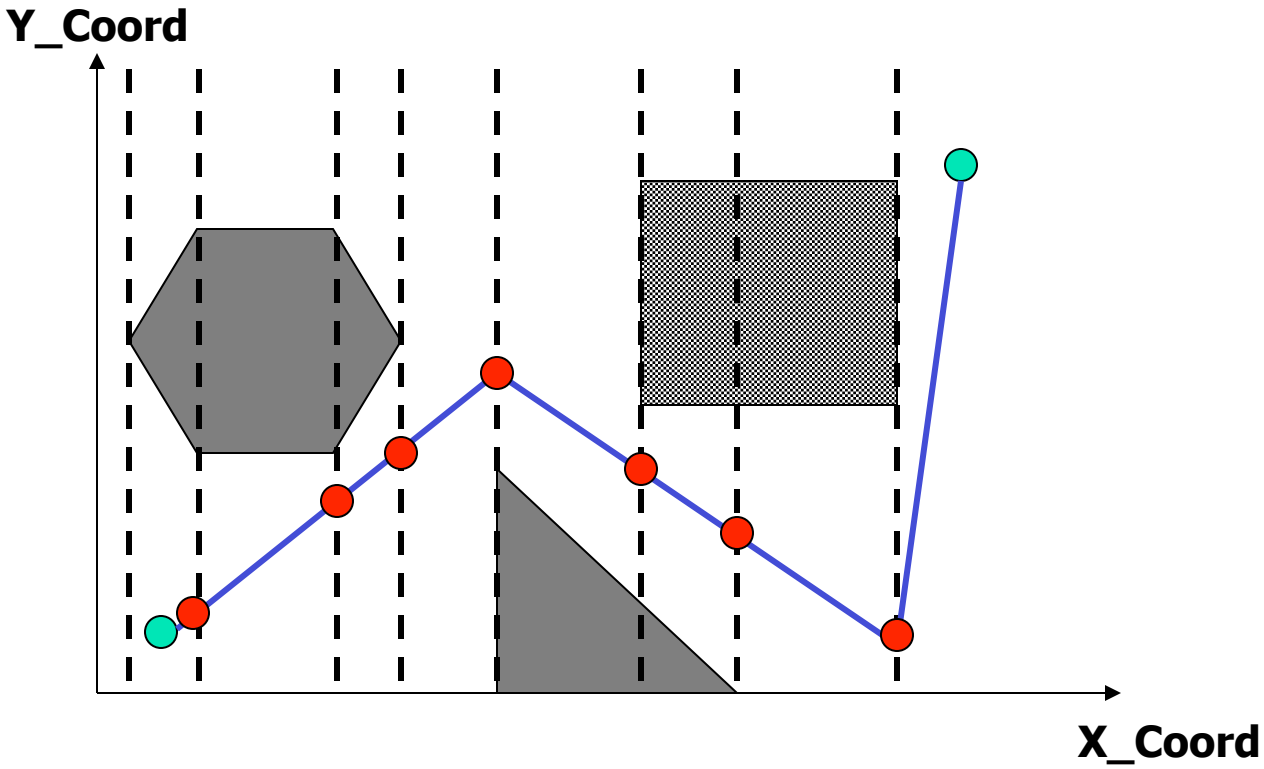
- One PSO implementation to solve this problem is found in [4].

RMP

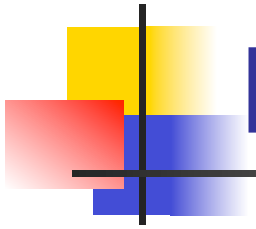


RMP

The idea is to optimize the y-coordinates at which the path intersects with the vertical lines



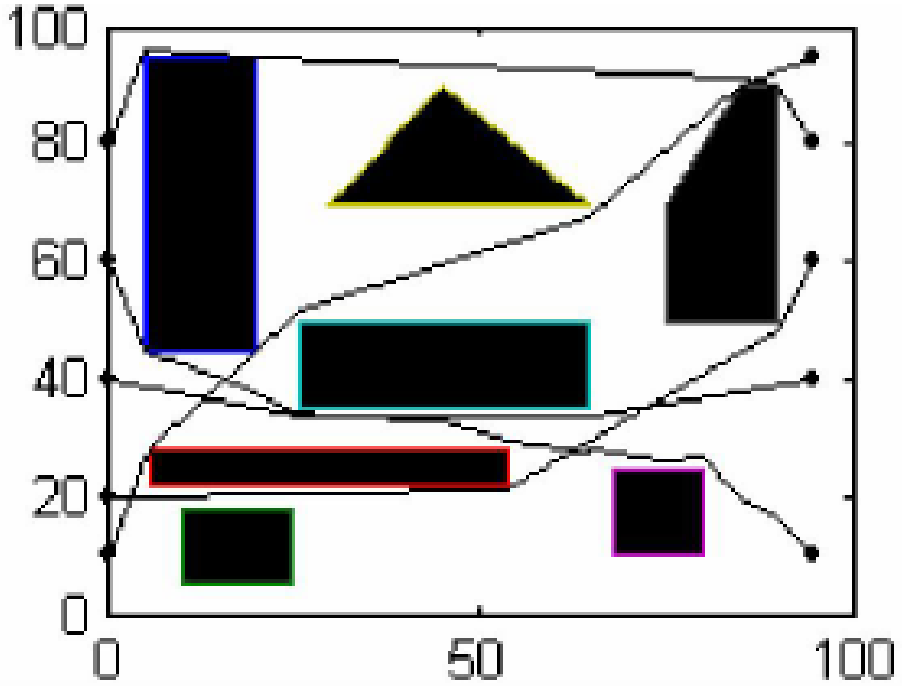
Vertical lines are drawn at the obstacles vertices



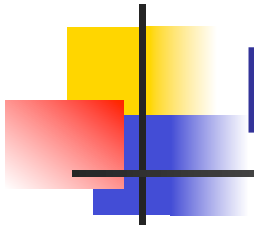
RMP

- How is the problem formulated?
  - PSO could be used to optimize the y-coordinates, each particle will be a n-dimensional vector, where n is the number of vertical lines.
  - The velocity is a n-dimensional vector of continuous values,
  - The objective could be to:
    - Minimize the path length [4],
    - Maximize the distance to obstacles,
    - A combination of both.

RMP



Five optimized paths using PSO [4]

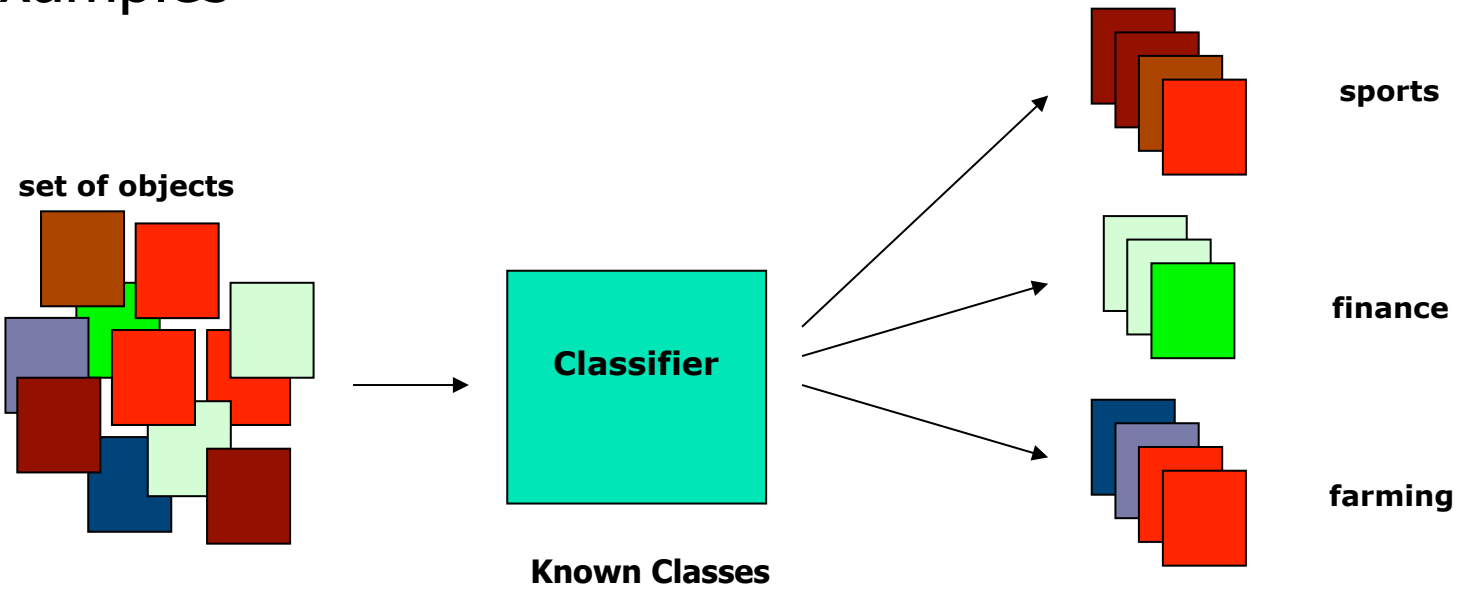


RMP

Start	Goal	Actual optimum	Obtained solution
0, 10	95, 95	140.45	143.54
0, 20	95, 60	111.02	112.86
0, 40	95, 40	96.47	98.41
0, 60	95, 10	117.40	119.47
0, 80	95, 80	113.35	115.75

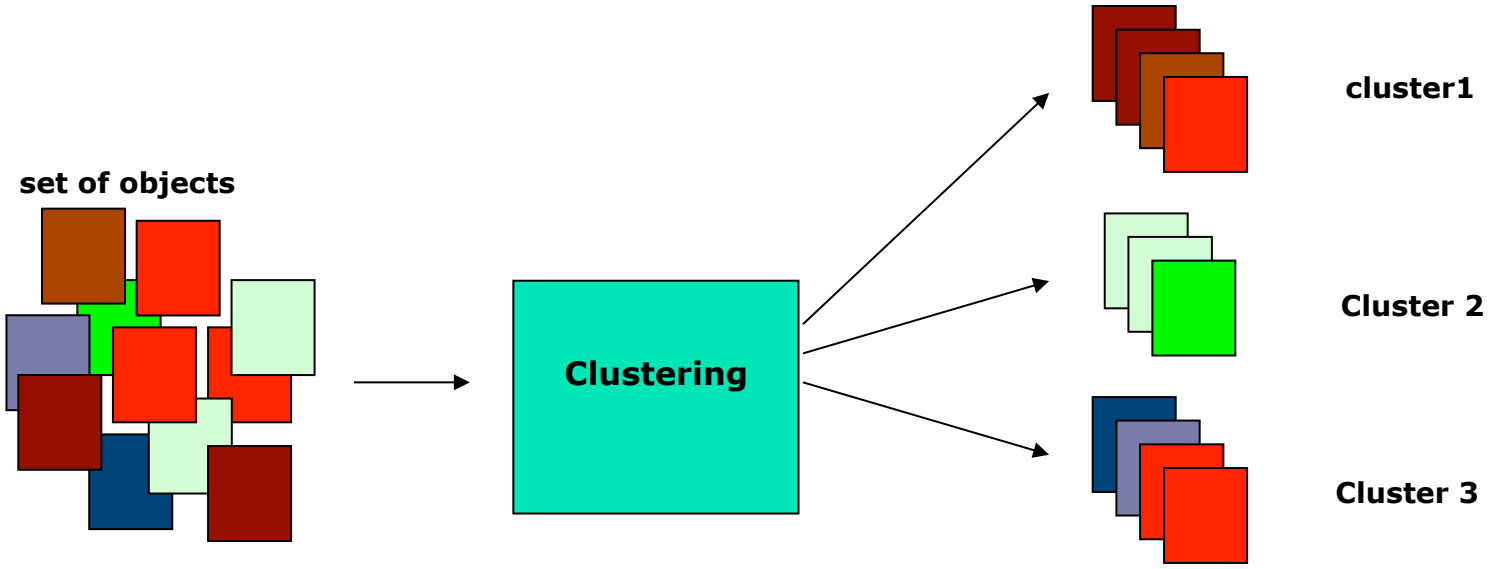
Classification

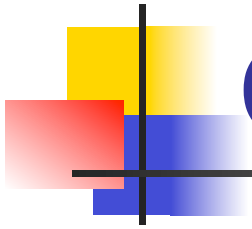
- Function that assigns an object to a class
- Infer that "object X is about sports"
- Automatically learn the function from a set of examples



Clustering

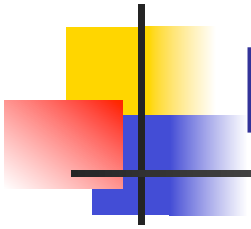
- Group similar objects into classes
- Similarity is high within the class and low between classes





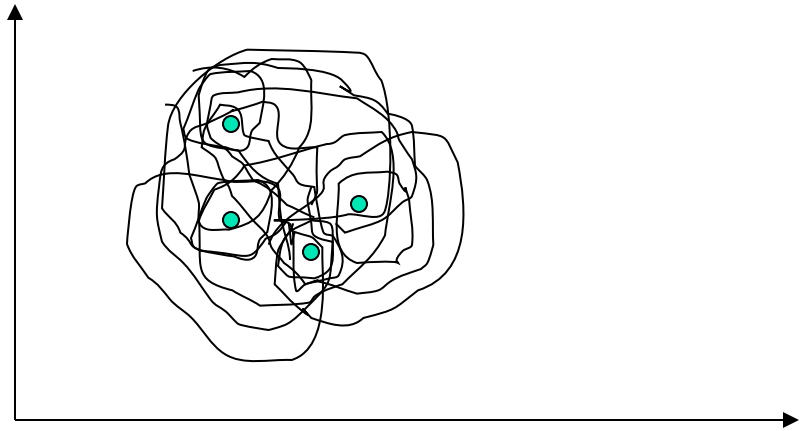
Clustering vs. Classification

- **Clustering**
  - Unsupervised
  - Uses unlabeled data
  - Organize patterns w.r.t. an optimization criteria
  - Notion of similarity
  - Hard to evaluate
  - Example: K-means, Fuzzy C-means, Hierarchical
- **Classification**
  - Supervised
  - Uses labeled data
  - Requires training phase
  - Domain sensitive
  - Easy to evaluate
  - Examples: Bayesian, k-NN, Decision Trees



Data Clustering Problem

Given *n* objects (each could be a vector of d features), group them in *k* groups (Clusters) in such a way that all objects in a single group have a "natural" relation to one another, and objects not in the same group are somehow different



Exponential Growth

- Number of possible partitions is given by

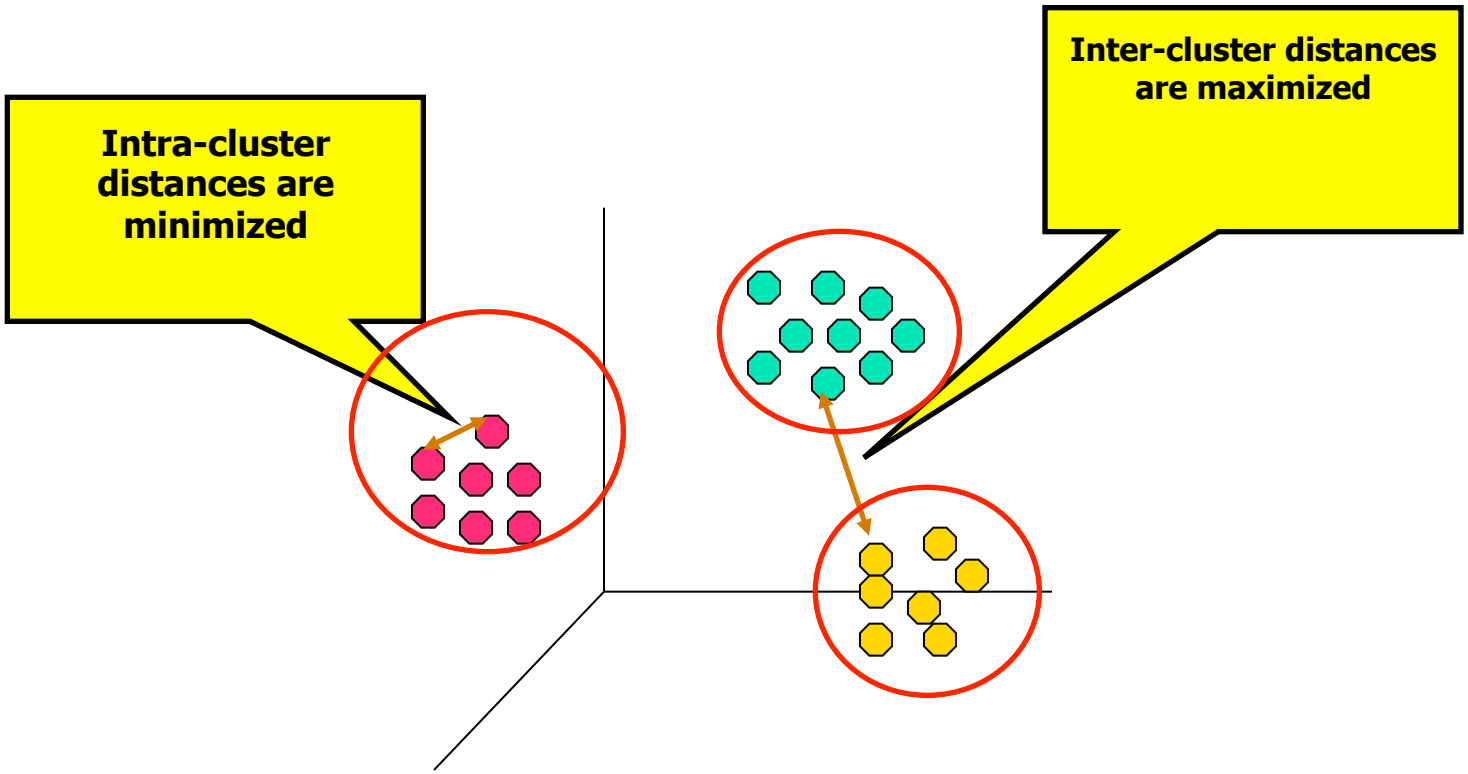
$$N(n,c) = \left(\frac{1}{c!}\right) \sum_{m=1}^{c} (-1)^{c-m} \binom{c}{m} m^n$$

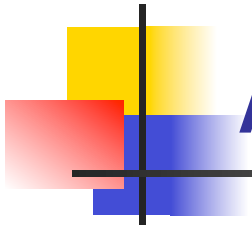
For example if *n=50 and c=4,* the number is $5.3x10^{28}$. If *n* is increased to 100, the number becomes $6.7x10^{58}$.

So enumerating all possible partitions for large problems is practically not feasible

Clustering

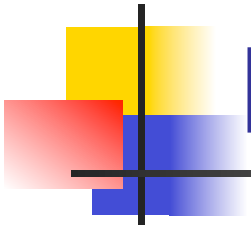
- Criteria





Applications of data clustering

- Data analysis
- Bioinformatics data
- Image segmentation
- GIS and spatial data
- Data and web mining
- Medical data
- Economic and financial data



K- means clustering Algorithm

Step 1: Initialize k Cluster centers

Repeat

Step 2: Distribute patterns among clusters using similarity measure and satisfying performance index.

Step 3: Compute new cluster centers

until no change in centers.

Apply the algorithm a number of times with different initial conditions

then choose the best solution.

K-means

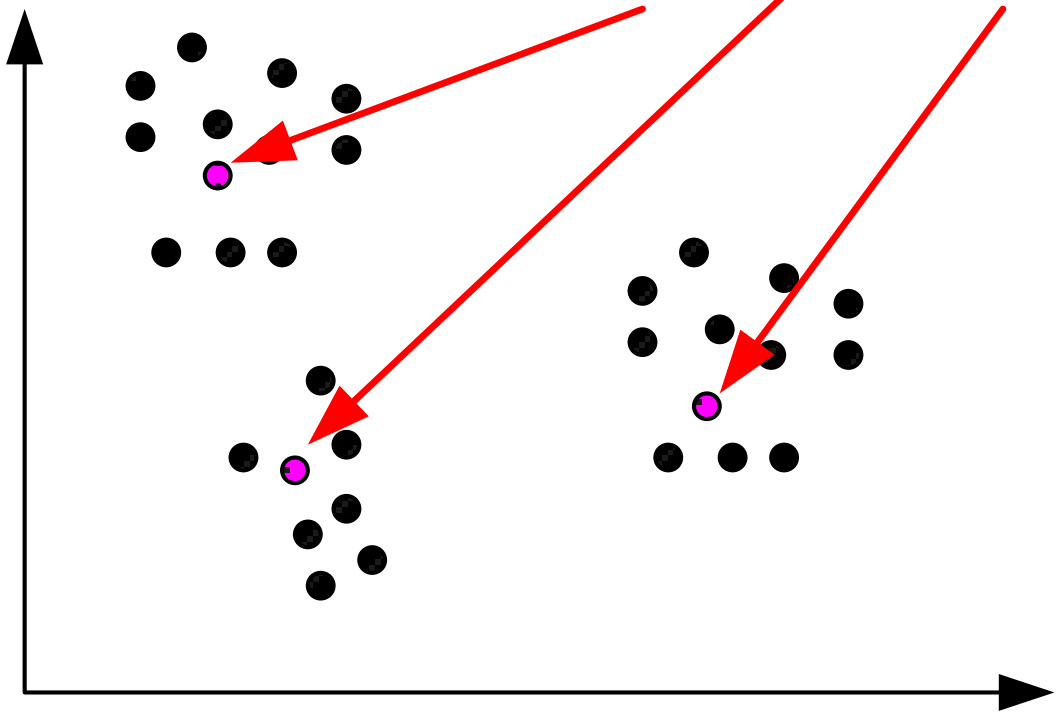
The similarity measure could be the inverse of the Euclidean distance and a performance index can be the dispersion

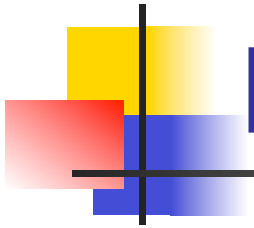
$$f(C_1, C_2, \ldots, C_k) = \sum_{i=1}^{k} \sum_{x_j \in C_i} \left\| x_j - z_i \right\|^2$$

Single Swarm Clustering

Particle representation

$$x^i = (m^{(1)}, m^{(2)}, m^{(3)})^i$$





PSO-Clustering

The PSO Clustering Algorithm

Initialize each particle with K random cluster centers.

  for iteration count = 1 to maximum iterations do

    for all particle i do

      for all pattern X*p* in the dataset do

        calculate Euclidean distance of X*p* with all cluster centroids

        assign X*p* to the cluster that have nearest centroid to X*p*

      end for

      calculate the objective function for the current centers and assignment

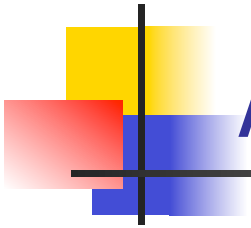
    end for

    find the personal best and global best position of each particle.

    Update the cluster centroids according to velocity updating and coordinate updating formula of PSO

  end for.

From [5]



Application to Document Clustering

- Problems
  - Overwhelming amount of text information.
  - Search engines return linear ranked list of documents.
  - Documents are of various topics yet interleaved.
- Motivation
  - Uncovering inherent groupings in large document sets.
  - Easy navigation through large document sets.
  - Easy navigation of search engine results.

Document Representation Model

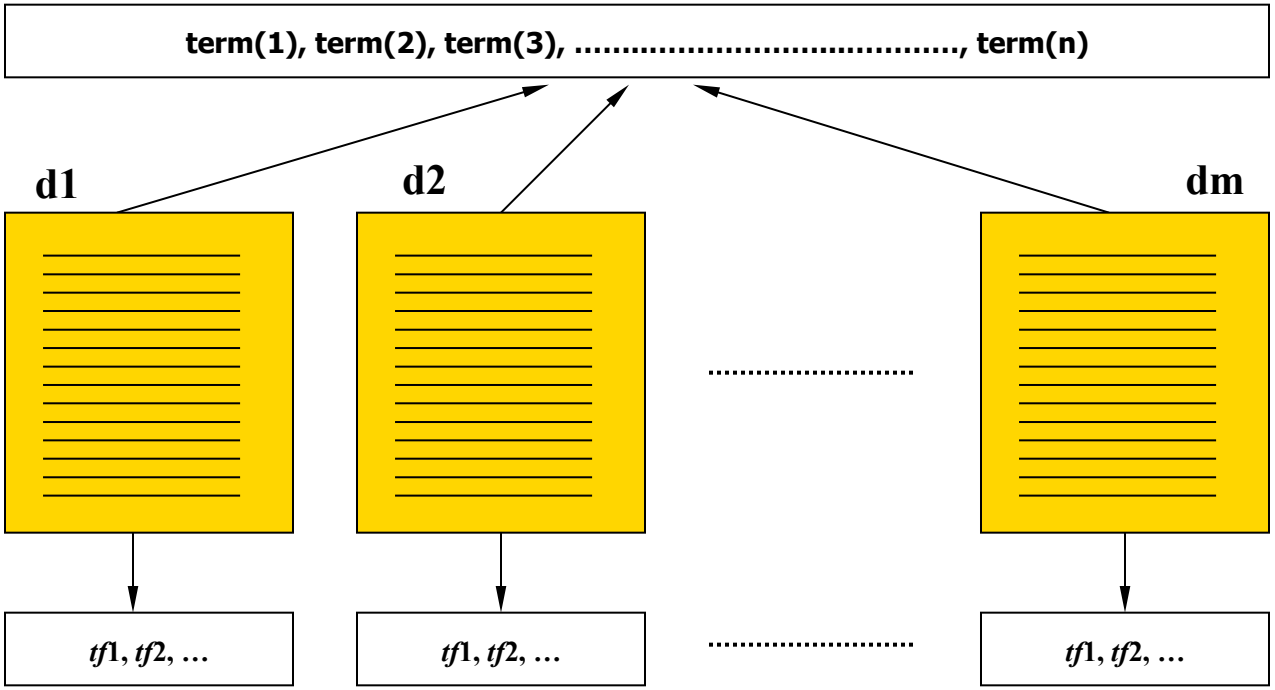
- Most Commonly used:
  Vector Space Model

- Document Representation:

$$\mathbf{d} = \{tf_1, tf_2, \ldots, tf_n\}$$

where $tf_i, i = 1, \ldots, n$ is the term frequency

Document Representation Model

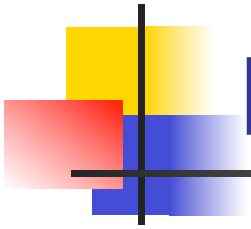


Term Frequency Vectors

Document Representation Model

- Term-Document matrix

	t_1	t_2	t_3	.....	t_n
$\mathbf{d}_1$	0.3	0	0.5	.....	0
$\mathbf{d}_2$	0	0.1	0.4	.....	0.2
:	:	:	:	.....	:
:	:	:	:	.....	:
:	:	:	:	.....	:
$\mathbf{d}_m$	0.1	0	0	.....	0



Document Representation Model

- Term Frequency Disadvantage:
  A term appearing frequently in all documents has less discriminating power.
- Document Frequency (DF):
  The number of documents in which a certain term appears.

- The less the DF of a term, the more discriminating power it has.

Document Representation Model

- TF-IDF
  TF is combined with the Inverse DF to form **Term Weight:**

$$w_i = tf_i \times \log(N / df)_i$$

- High TF, Low DF → High TW
  (i.e. high discrimination power)

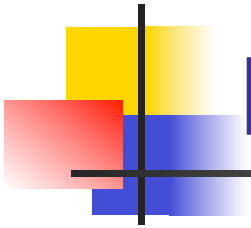
Document Representation Model

- Normalized *tf-idf*
  document length is taken into consideration

$$w_i = \frac{tf_i \times \log(N / df_i)}{\sqrt{\sum_j \left[tf_j \times \log(N / df_j)\right]^2}}$$

- Normalized $\log(tf)$-*idf*
  reduce the effect of large differences in term frequencies

$$w_i = \frac{\log(tf_i + 1) \times \log(N / df_i)}{\sqrt{\sum_j \left[\log(tf_j + 1) \times \log(N / df_j)\right]^2}}$$



Document Representation Model

- Preprocessing
  - Stop-word Removal
    (very common words like “the”, “and”)
  - Stemming
    (e.g. computer, computing → comput)
- Feature Selection
  - Information Gain (IG)
  - Mutual Information (MI)
  - Document Frequency (DF)
  - CHI^2 Statistic

Similarity Measure

- Cosine Measure (most commonly used)

$$\cos(\mathbf{x}, \mathbf{y}) = \frac{\mathbf{x} \cdot \mathbf{y}}{\|\mathbf{x}\| \|\mathbf{y}\|}$$

- Euclidean Distance

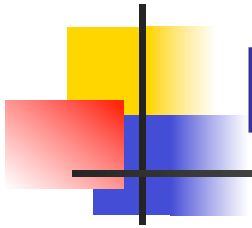
$$\|\mathbf{x} - \mathbf{y}\|_2$$

Performance measure

- The average distance of documents to the cluster centroid (ADDC)

$$(1/k)\sum_{i=1}^{k}\sum_{j=1}^{n_i}\left\|d_{ij}-m_i\right\|_2/n_i$$

n_i is number of documents in cluster i



Hybrid PSO-K-means

(1) Start the PSO clustering process until the maximum number of iterations is exceeded

(2) Inherit clustering result from PSO as the initial centroid vectors of K-means module.

(3) Start K-means process until maximum number of iterations is reached.

Experiment From [6]

Data	no. of document	no. of terms	no. of clusters
Dataset1	414	6429	9
Dataset2	313	5804	8
Dataset3	204	5832	6
Dataset4	878	7454	10

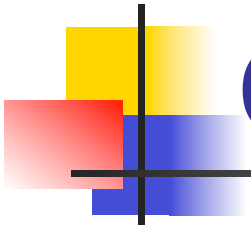
Comparison between K-means, PSO, Hybrid PSO

		ADDC value		
		K-means	PSO	Hybrid PSO
Dataset1	Euclidian	8.17817	8.11009	6.38039
	Cosine	8.96442	10.41271	8.14551
Dataset2	Euclidian	7.26175	6.25172	4.51753
	Cosine	8.07653	9.57786	7.21153
Dataset3	Euclidian	4.59539	4.14896	2.25961
	Cosine	4.97171	5.71146	4.00555
Dataset4	Euclidian	9.08759	8.62794	6.37872
	Cosine	10.1739	12.8927	9.5379

For PSO ***w*** is initially set as 0.72 and the acceleration coefficient constants ***c1*** and ***c2*** are set as 1.49.
w is reduced by 1% at each generation to ensure good convergence.

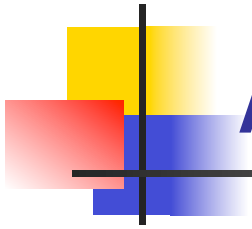
References

1. El-Abd, M., Hassan, H., Anis, M., Kamel, M. and El-Masry, M. I., "Discrete Cooperative Particle Swarm Optimization for FPGA Placement", Applied Soft Computing, Vol. 10, No. 1, pp. 284-295, January 2010.
2. Hassan, H, VLSI Placement , Presentation, 2009
3. J. Kennedy and R. C. Eberhart. “Particle Swarm Optimization”. Proceedings of the IEEE International Conference on Neural Networks, vol. 4, pp. 1942–1948, 1995.
4. K. Lei, Y. Qiu and Y. He. “A Novel Path Planning for Mobile Robots Using Modified Particle Swarm Optimizer”. Proc. of the 1st International Symposium on Systems and Control in Aerospace and Astronautics ISSCAA, pp. 981-984,2006.
5. Abraham, A., Das, S., & Roy, S. Swarm intelligence algorithms for data clustering. In Soft computing for knowledge discovery and data mining Part IV, pp. 279-313, 2007.
6. X. Cui; J. Gao and T. E. Potok. A flocking based algorithm for document clustering analysis. *Journal of Systems Architecture*, 52(8-9):505–515, August-September 2006
7. A. Ahmadi; F. Karray and M. Kamel. Multiple cooperating swarms for data clustering. In *IEEE Swarm Intelligence Symposium*, pages 206–212, 2007.



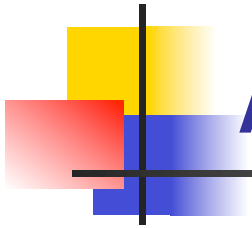
Outline

- Introduction,
- PSO,
- Discrete PSO,
  - Binary PSO
  - Permutation PSO,
- Applications,
- Adaptation
- Cooperation,
- References.



Adaptation

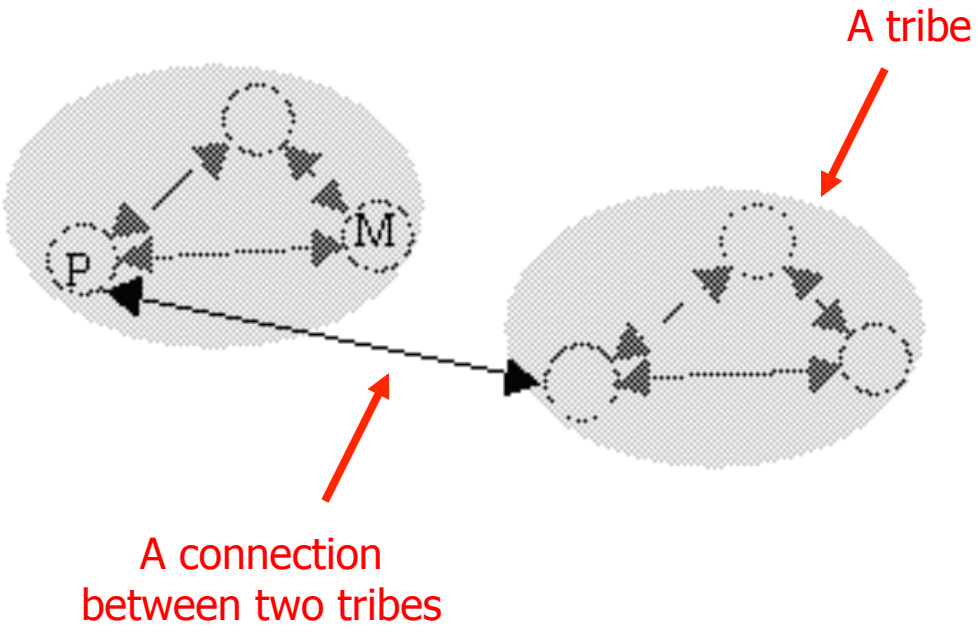
- An adaptive parameter free PSO, referred to as *TRIBES*, was proposed in [17],
- The purpose was to have the user to call PSO algorithm without the need to set any parameters,
- The main point was to adapt the number particles used in the search.

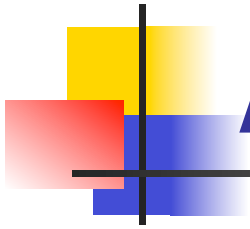


Adaptation

- A *tribe* refers to a group of connected particles,
- All the tribes should have some type of connection between them to inform one another of their findings,
- This will help in deciding which is the global minimum among all the different solutions that was found by the different tribes.

Adaptation





Adaptation

- Definitions:
  - A *good* particle is a particle that has its pbest improved in the last iteration, otherwise it's *neutral*,
  - Each particle memorizes the last two performance variations, a particle with both variations as improvements is an *excellent* particle,
  - *G* is the number of good particles in a tribe.

Adaptation

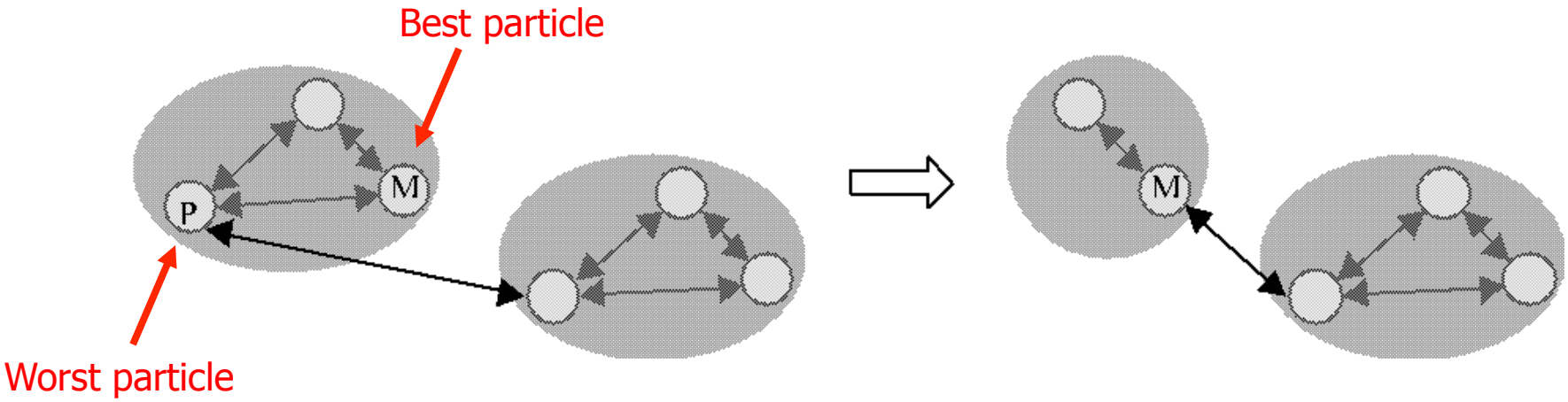
- Definitions:
  - A tribe is marked as good depending on the value of *G*, if the total number of particles in a tribe is *T* :

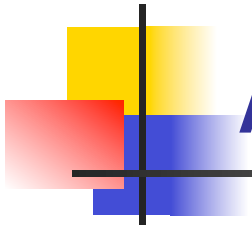
$$r = U(0,1)$$

$$Tribe = \begin{cases} good, & r < \frac{G}{T} \\ bad, & otherwise \end{cases}$$

Adaptation

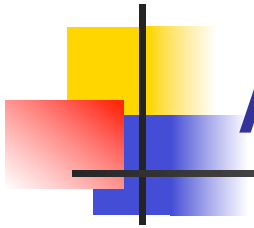
- A good tribe deletes its worst particle to conserve the number of performed function evaluations.





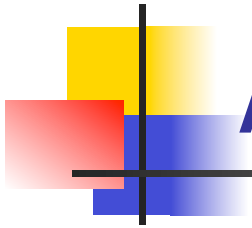
Adaptation

- On the other hand, every bad tribe generates a new random particle simultaneously,
- All the new particles form a new tribe,
- Each particle gets connected to the tribe that generated it through its best particle.



Adaptation

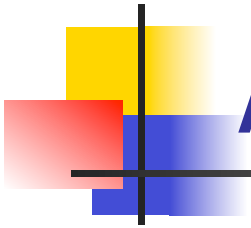
- The idea is to start with a single particle,
- Most likely, this particle won't improve in the first iteration. Hence, it will generate another particle forming another tribe,
- If both don't improve, they will simultaneously generate two other particles forming a third two-particle tribe,
- And the process continues…



Adaptation

- If things go bad, larger and larger tribes will be generated to increase the swarm search power,

- On the other hand, if good solutions start to occur, good tribes will start removing its worst particles reducing the tribes size possibly to complete extinction.



Adaptation

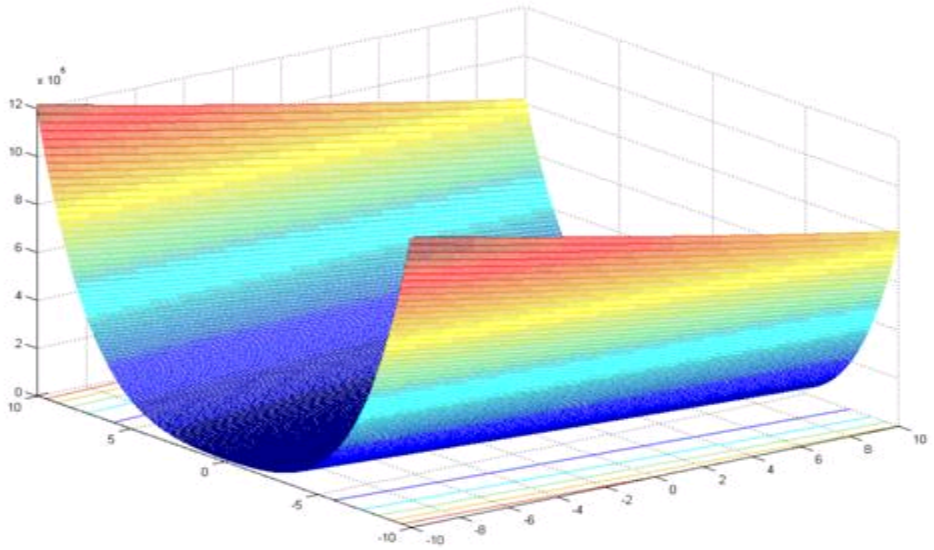
- Assume that one is constrained with only 40000 function evaluations to perform, what would be the optimal swarm size to use in order to find the best result?

- The question was answered by solving the Rosenbrock function in a dimensionality of 30.

2D Rosenbrock

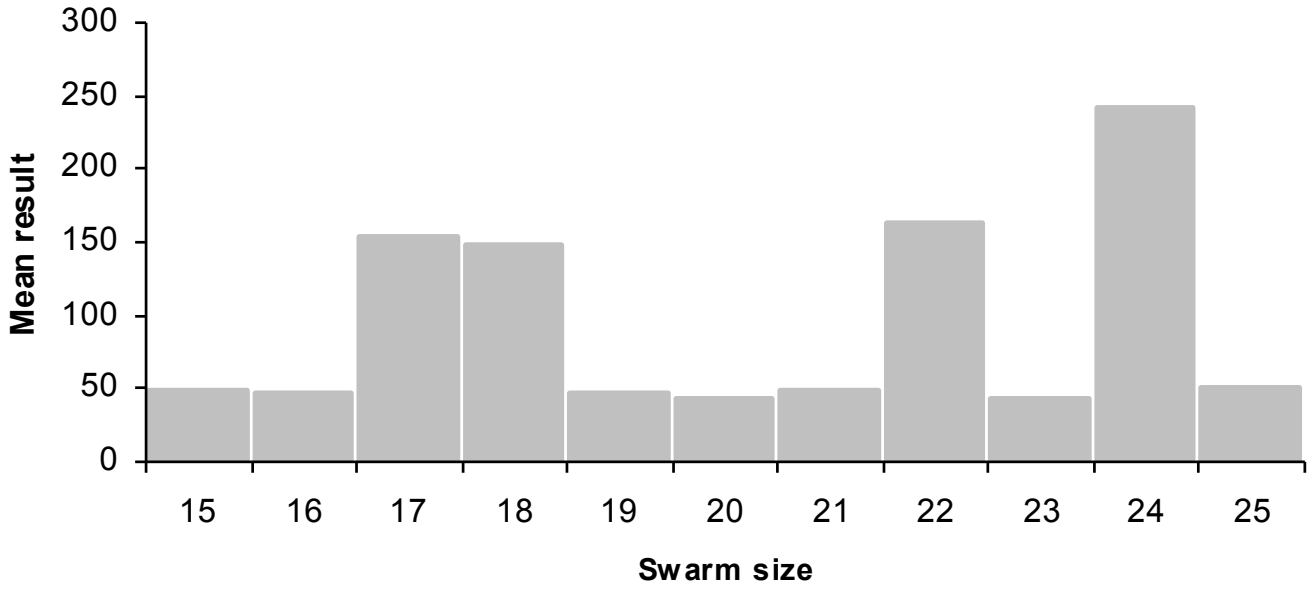
- Rosenbrock:

$$f(x) = \sum_{i=1}^{d-1} \left[(1 - x_i)^2 + 100 \left(x_{i+1} - x_i^2 \right)^2 \right]$$



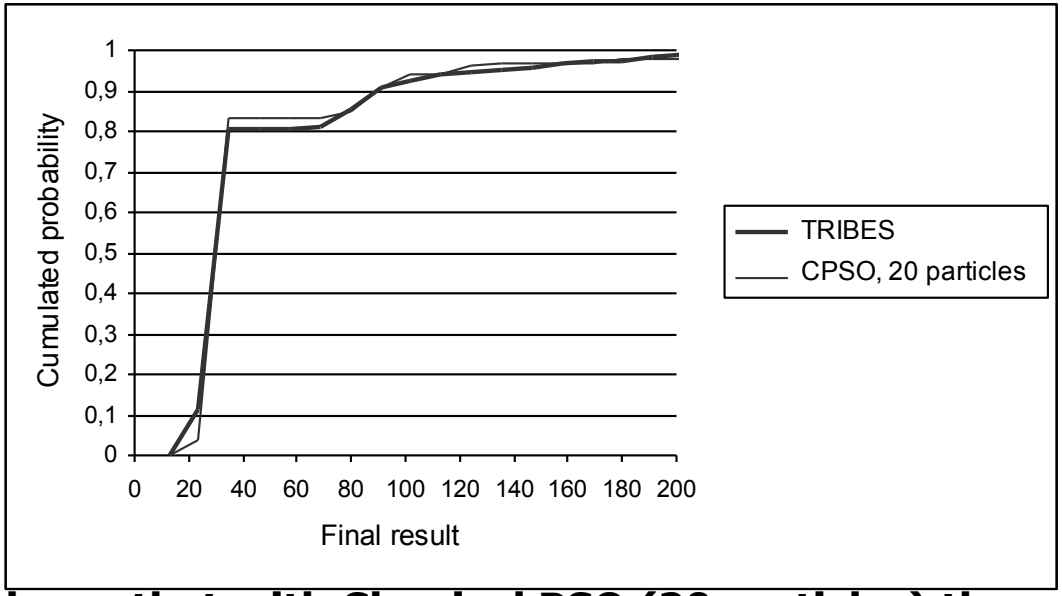
Adaptation

- The results show that using 20 particles is the best. However, there's no clear relation between the number of particles and the performance.

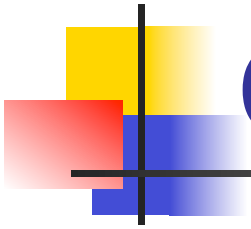


Adaptation

- The function was optimized using TRIBES and it shows that it was able to exactly follow the behaviour of a swarm of 20 particles, 500 runs, 40000 function eval.

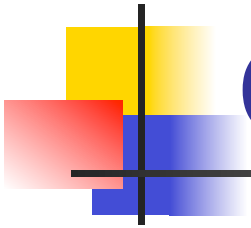


the graph shows that with Classical PSO (20 particles),there is 83% chance of obtaining a value smaller than 40, against 80 % with TRIBES. For small values, TRIBES is clearly better: 11.5 % chance to obtain a value smaller than 25, against 4 % with CPSO



Outline

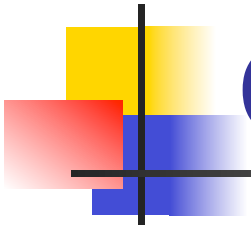
- Introduction,
- PSO,
- Discrete PSO,
  - Binary PSO
  - Permutation PSO,
- Applications,
- Adaptation
- Cooperation,
- References.



Cooperation

- Different approaches have been proposed to introduce cooperation among different swarms,

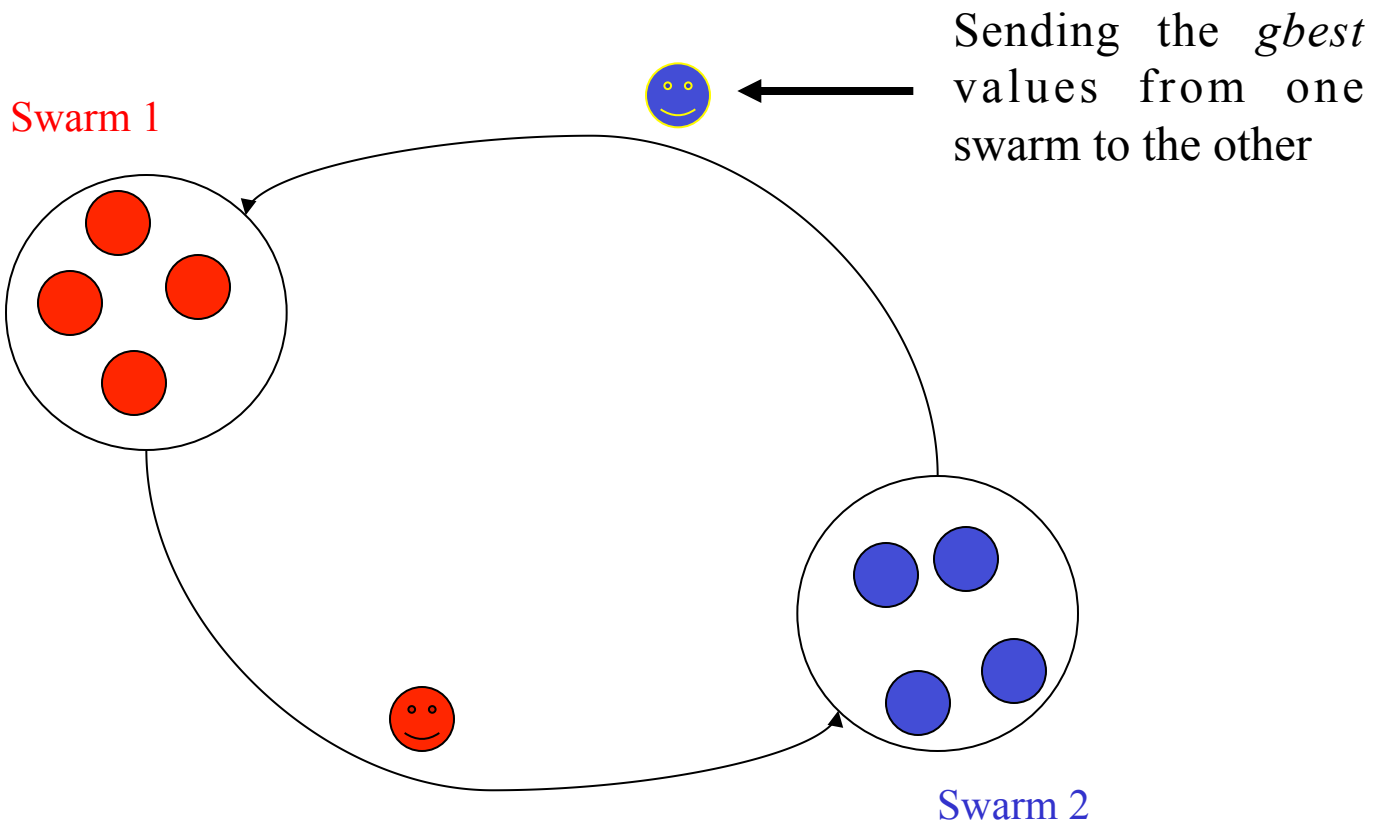
- These approaches include:
  - Concurrent PSO,
  - Cooperative PSO,
  - Hybrid Cooperative PSO.

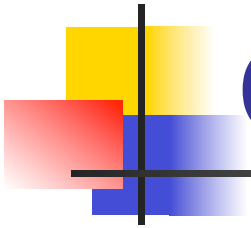


Cooperation

- One form of multi-swarm algorithms is known as *concurrent-PSO* [13],
- Two different swarms are updated in parallel, both using different algorithms,
- The swarms exchanged their *gbest* values every pre-determined number of iterations,
- Both swarms are to track the better *gbest*.

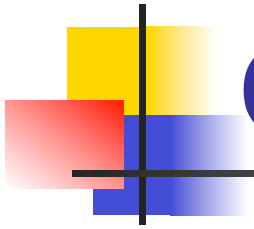
Cooperation





Cooperation

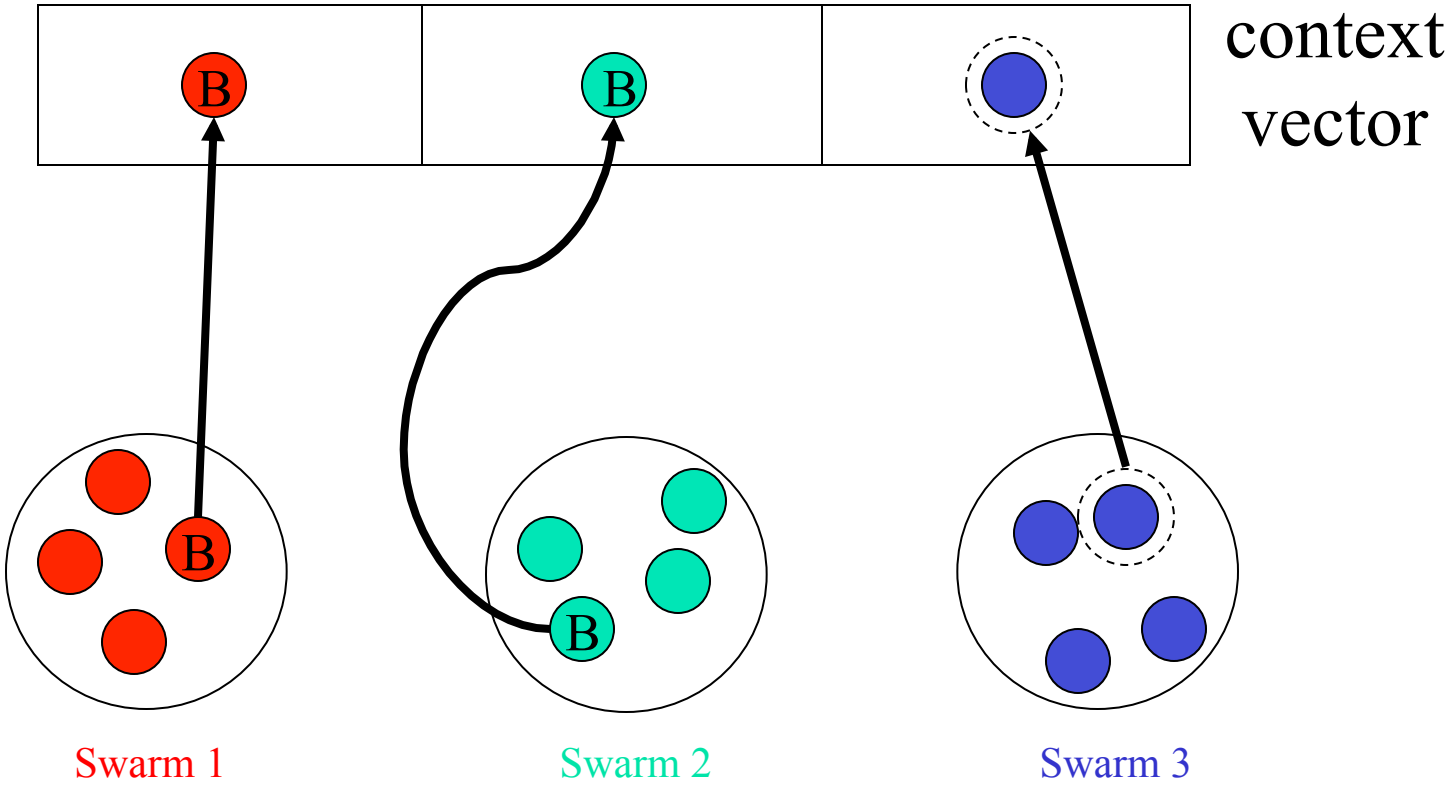
- A method referred to as *Cooperative Particle Swarm Optimizers (CPSO)* was proposed by van den Bergh et. al. in 2004 [14],
- Applies the same concept of the cooperative GA approach,
- Having multiple swarms where the fitness of any particle in any swarm depends on the particles of the other swarms.

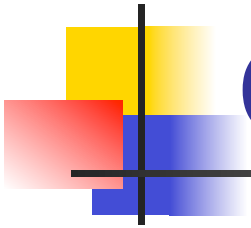


Cooperation

- The general idea is to have different swarms optimizing different variables of the problem (different dimensions of the solution),
- The fitness of any particle is determined by its value and the value of the best particles in all the other swarms,
- Performs best if the problem variables are independent.

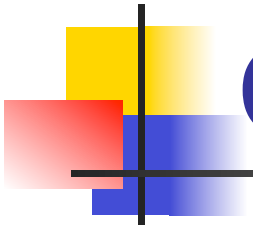
Cooperation





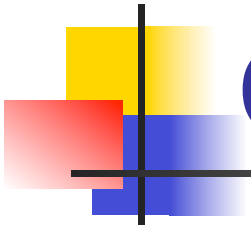
Cooperation

- A different approach is referred to as *Hybrid Cooperative PSO (CPSO_H)* was also proposed by van den Bergh et. al. in 2004 [14],
- Two swarms were serially updated,
- One swarm using the normal PSO algorithm and the other swarm is using CPSO.



Cooperation

- Each swarm is updated for one iteration only,
- When the PSO swarm get updated, its *gbest* values is sent to the CPSO swarm,
- The CPSO swarm uses the elements of the received *gbest* to update random particles of its sub-swarms.

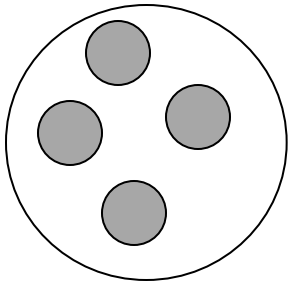


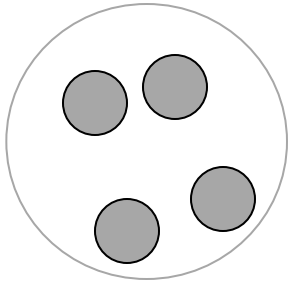
Cooperation

- After CPSO gets updated, it sends its context vector back to the PSO swarm,
- The PSO swarm uses the received context vector to replace a randomly chosen particle.

Cooperation

Swarm 1

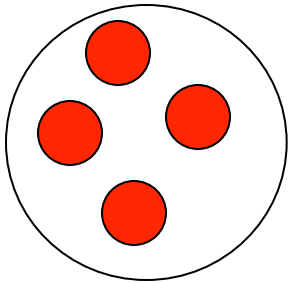


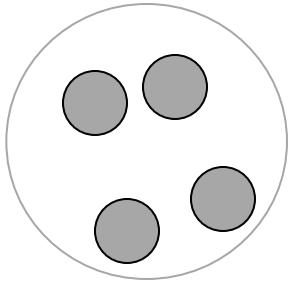


Swarm 2

Cooperation

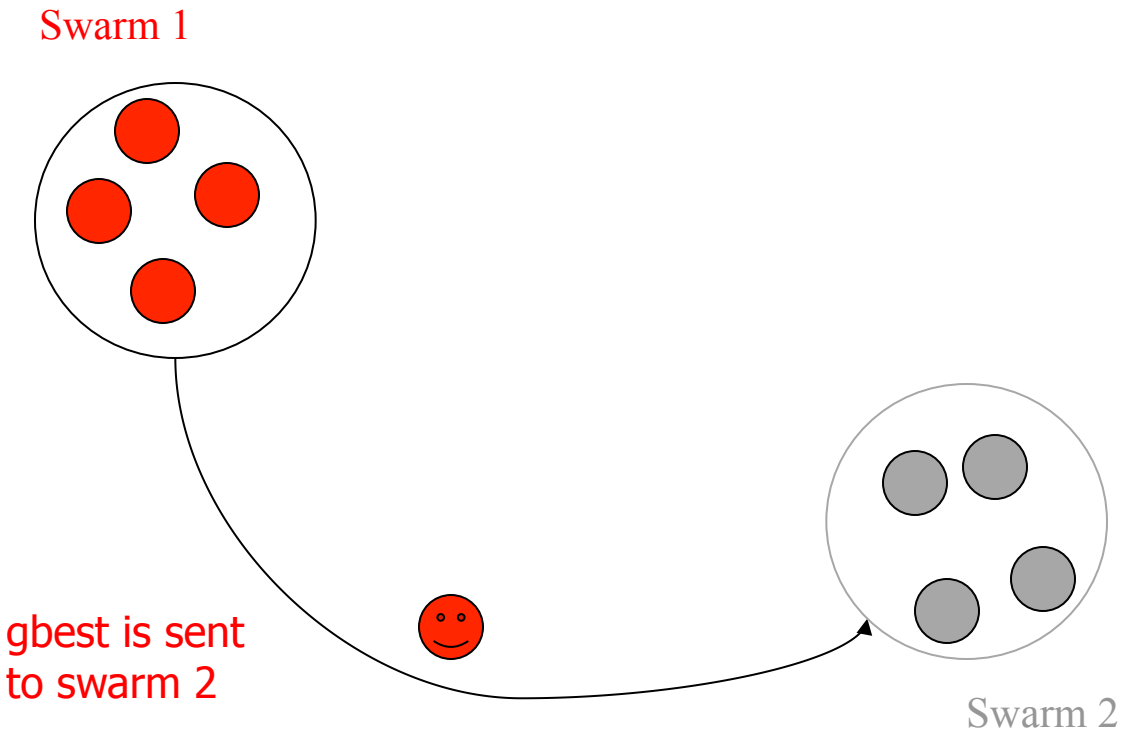
Swarm 1
gets updated



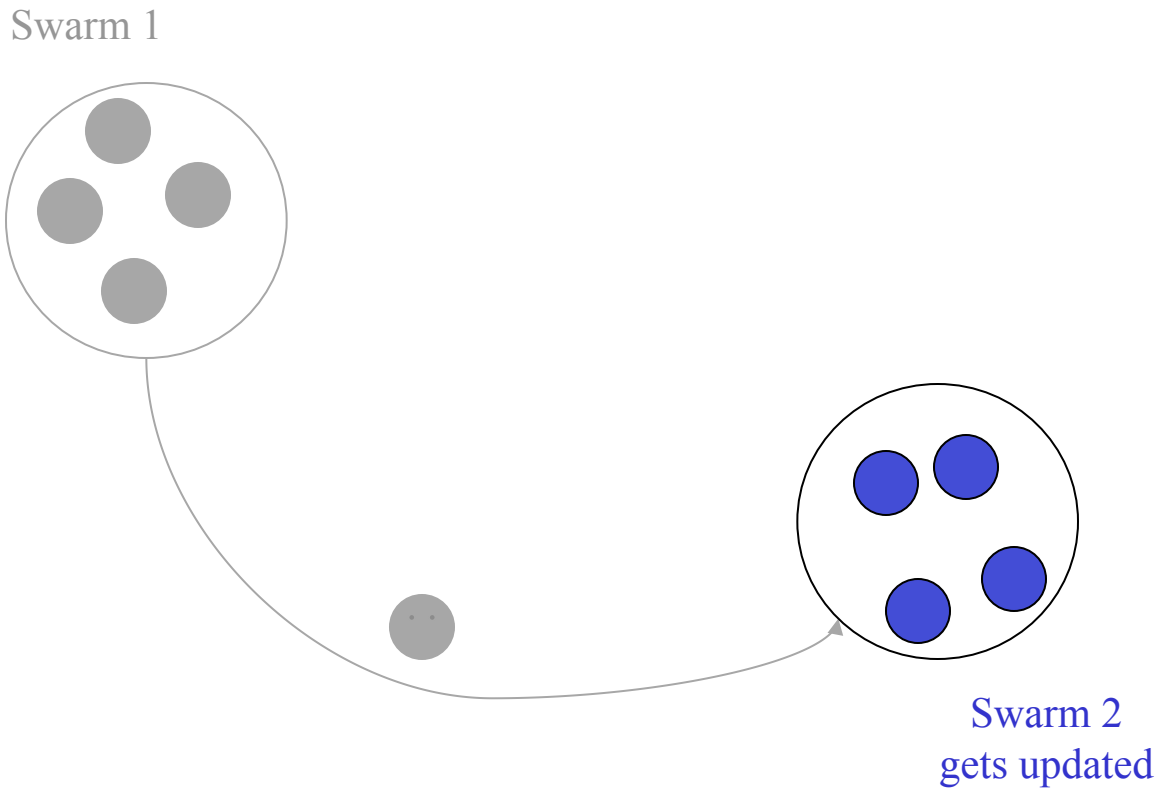


Swarm 2

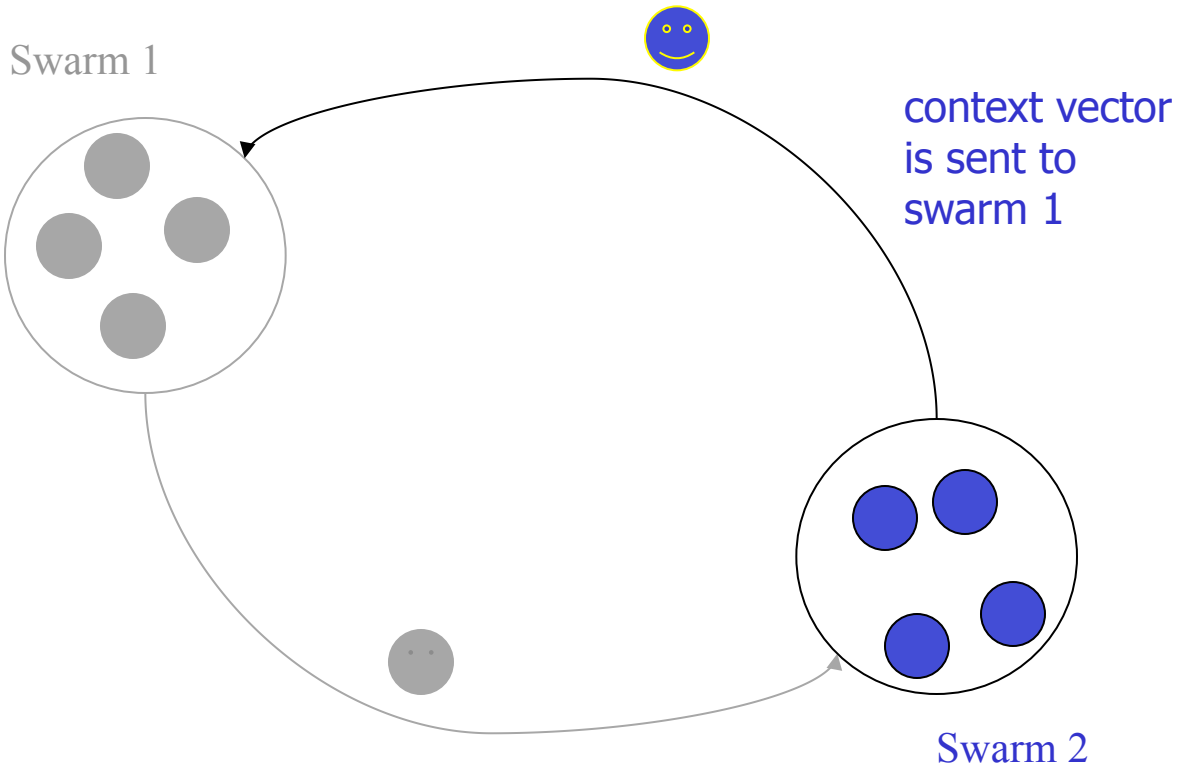
Cooperation



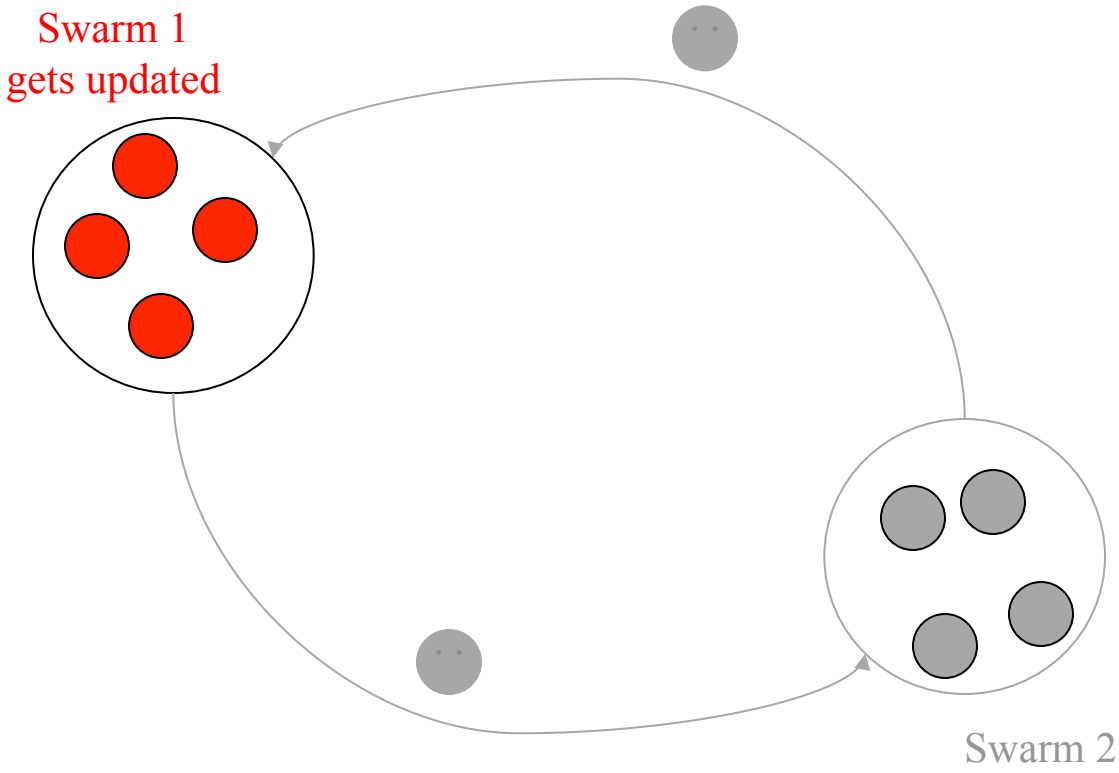
Cooperation



Cooperation



Cooperation



References

1. C. W. Reynolds.”Flocks, Herds and Schools: A Distributed Behavioural Model”. Computer Graphics, Vol. 21, Number 4, pp. 25-34, 1987.
2. C. W. Reynolds. Homepage on Boids. http://www.red3d.com/cwr/boids/
3. M. A. Montes de Oca. “Particle Swarm Optimization – Introduction”. Available at: http://iridia.ulb.ac.be/%7Emmontes/slidesCIL/slides.pdf
4. F. Heppner and U. Grenander, “A Stochastic Nonlinear Model for Coordinated Bird Flocks”, In S. Krasner, (ed), The Ubiquity of Chaos. AAAS pub., 1990.
5. J. Kennedy and R. C. Eberhart. “Particle Swarm Optimization”. Proceedings of the IEEE International Conference on Neural Networks, vol. 4, pp. 1942–1948, 1995.
6. R. C. Eberhart and J. Kennedy. “A New Optimizer using Particle Swarm Theory,” Proceedings of the 6th International Symposium on Micro Machine and Human Science, pp. 39–43, 1995.

References

7. I. C. Trelea. ”The Particle Swarm Optimization Algorithm: Convergence Analysis and Parameter Selection”. Information Processing Letter, vol. 85, no. 6, pp. 317-325, 2003.
8. M. Clerc and J. Kennedy. “The Particle Swarm Explosion, Stability and Convergence in a Multi-dimensional Complex Space,” IEEE Transactions on Evolutionary Computation, vol. 6, no. 1, pp. 58–73, 2002.
9. R. C. Eberhart, P. Simpson, and R. Dobbins, *Computational Intelligence*. PC Tools: Academic, ch. 6, pp. 212–226, 1996.
10. J. Kennedy and R. Mendes. “Neighbourhood Topologies in Fully Informed and best-of-neighbourhood Particle Swarms”. IEEE Transactions on Systems, Man and Cybernetics – Part C. Vol. 36, Issue 4, pp. 515-519, 2006.
11. J.Kennedy and R. C. Eberhart. “A Discrete Binary Version of The Particle Swarm Algorithm”. Proceedings of the World Multiconference on Systemics, Cybernetics and Informatics, pp. 4101-4109, 1997.

References

12. M. Clerc, “Discrete Particle Swarm Optimization,” in New Optimization Techniques in Engineering. Springer-Verlag, 2004.
13. S. Baskar and P. N. Suganthan. “A Novel Concurrent Particle Swarm Optimizer”. Proceedings of the IEEE Congress on Evolutionary Computation, vol. 1, pp. 792-796, 2004.
14. F. van den Bergh and A. P. Engelbrecht, “A cooperative approach to Particle Swarm Optimization”. IEEE Transactions on Evolutionary Computing, vol. 8, no. 3, pp. 225– 239, 2004.
15. W. Pang, K. Wang, C. Zhou, L. Dong, M. Liu, H. Zhang and J. Wang. “Modified Particle Swarm Optimization Based On Space Transformation for Solving Travelling Salesman Problem”. Proceedings of the third international conference on Machine Learning and Cybernetics, pp. 26-29, 2004.
16. K. Lei, Y. Qiu and Y. He. “A Novel Path Planning for Mobile Robots Using Modified Particle Swarm Optimizer”. Proc. of the 1st International Symposium on Systems and Control in Aerospace and Astronautics ISSCAA, pp. 981-984,2006.
17. http://clerc.maurice.free.fr/pso/Tribes/Tribes_doc.zip