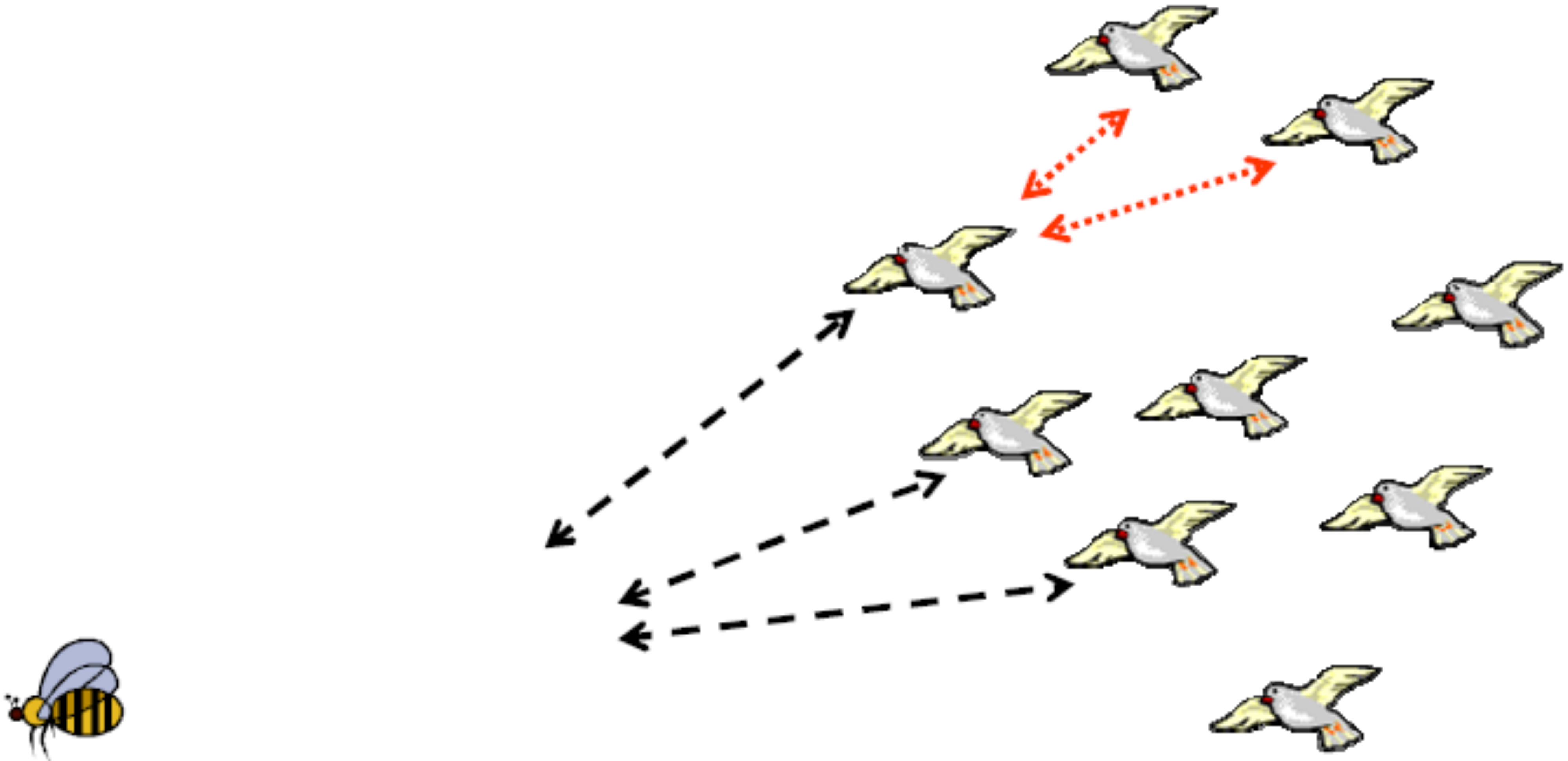
14 Particle Swarm Optimisation

Particle Swarm Optimisation (PSO)

- developed by Eberhart & Kennedy in 1995
- imitates the social behaviour of humans or animals
  - swarms of bees, flocks of birds, schools of fish
- searches for an optimal solution through the interactions of individuals
- individuals have memory and can learn
- they try to improve themselves by observing and imitating neighbours

Particle Swarm Optimisation (PSO)



- PSO algorithms are simple with low overhead
- so they are popular

PSO Algorithm

- each particle `i` evaluates the function `f` to maximise (or minimise) at each point in space that it visits
- it remembers personal information:
  - x_i : its current location
  - $f(p_i)$: the best value of the function it has found so far
  - p_i : the point in space where it was found
- it also knows about global (collective) progress:
  - $f(g)$: the globally best value that a member of the flock has found so far
  - g : the point in space where that optimal value was found

PSO Algorithm

- the particles repeatedly change their velocity based on a mixture of the personal and global information:

$$v_i = \omega . v_i + \varphi_p . r_p . (p_i - x_i) + \varphi_g . r_g . (g - x_i)$$

where ω, φ_p and φ_g are user-defined weightings

and r_p and r_g are random values in the interval [0,1]

- and hence they update their position:

$$x_i = x_i + v_i$$

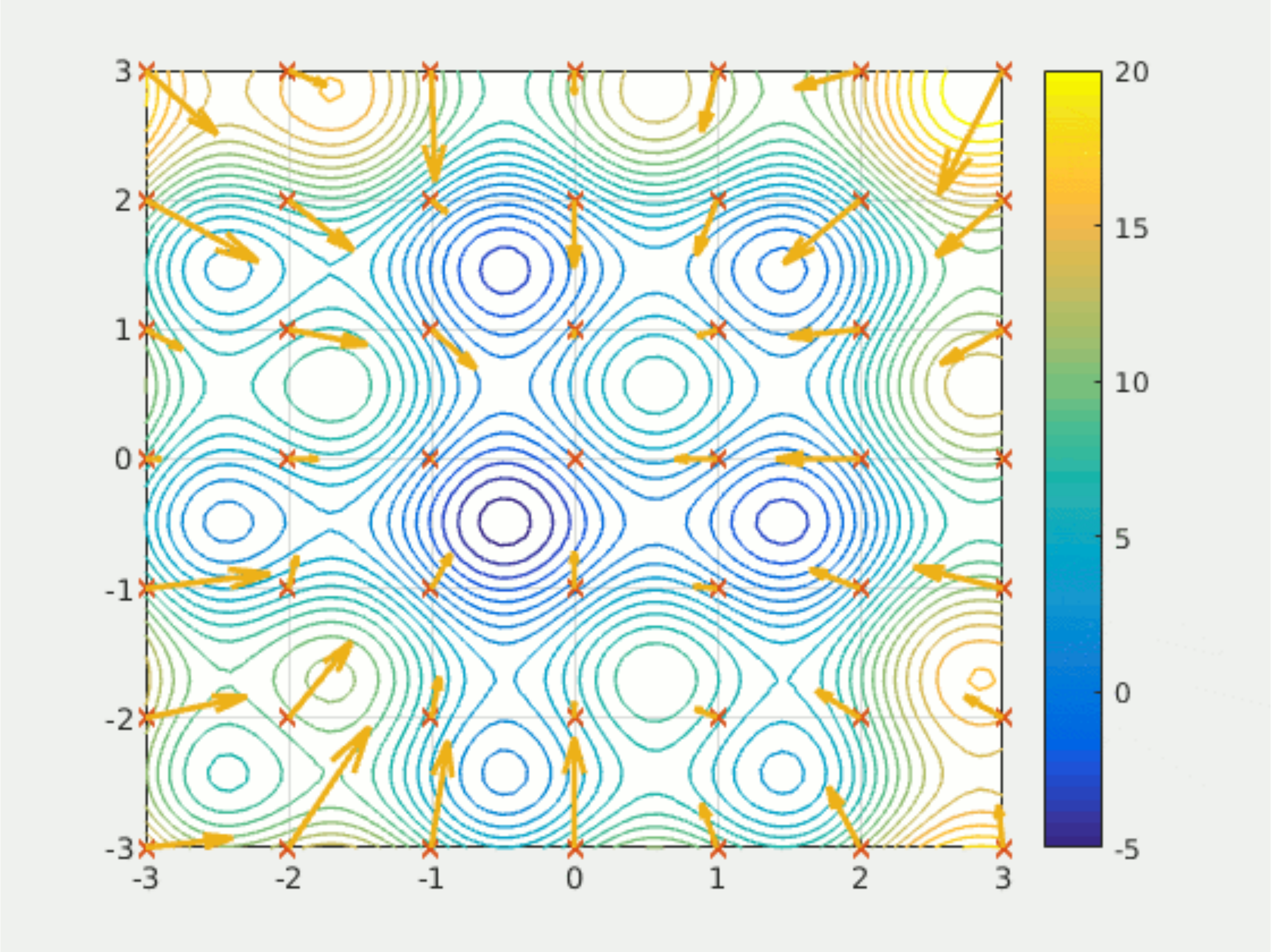
- this continues until the termination criterion is met

PSO Full Algorithm

```
let S = number of particles in the swarm
    g be the best known position of the entire swarm
    blo & bup be the upper and lower boundaries of the search space
    ω, φp, and φ are user-selected weighting parameters
    f is the cost function to minimise

for each particle i = 1, ..., S do

   Initialize the particle's position with a uniformly distributed random vector: xi ~ U(blo, bup)
   Initialize the particle's best known position to its initial position: pi ← xi

   if f(pi) < f(g) then
       update the swarm's best known  position: g ← pi

   Initialize the particle's velocity: vi ~ U(-|bup-blo|, |bup-blo|)

while a termination criterion is not met do:

   for each particle i = 1, ..., S do

      for each dimension d = 1, ..., n do
         Pick random numbers: rp, rg ~ U(0,1)
         Update the particle's velocity: vi,d ← ω vi,d + φp rp (pi,d-xi,d) + φg rg (gd-xi,d)

     Update the particle's position: xi ← xi + vi

      if f(xi) < f(pi) then
         Update the particle's best known position: pi ← xi
         if f(pi) < f(g) then
            Update the swarm's best known position: g ← pi
```

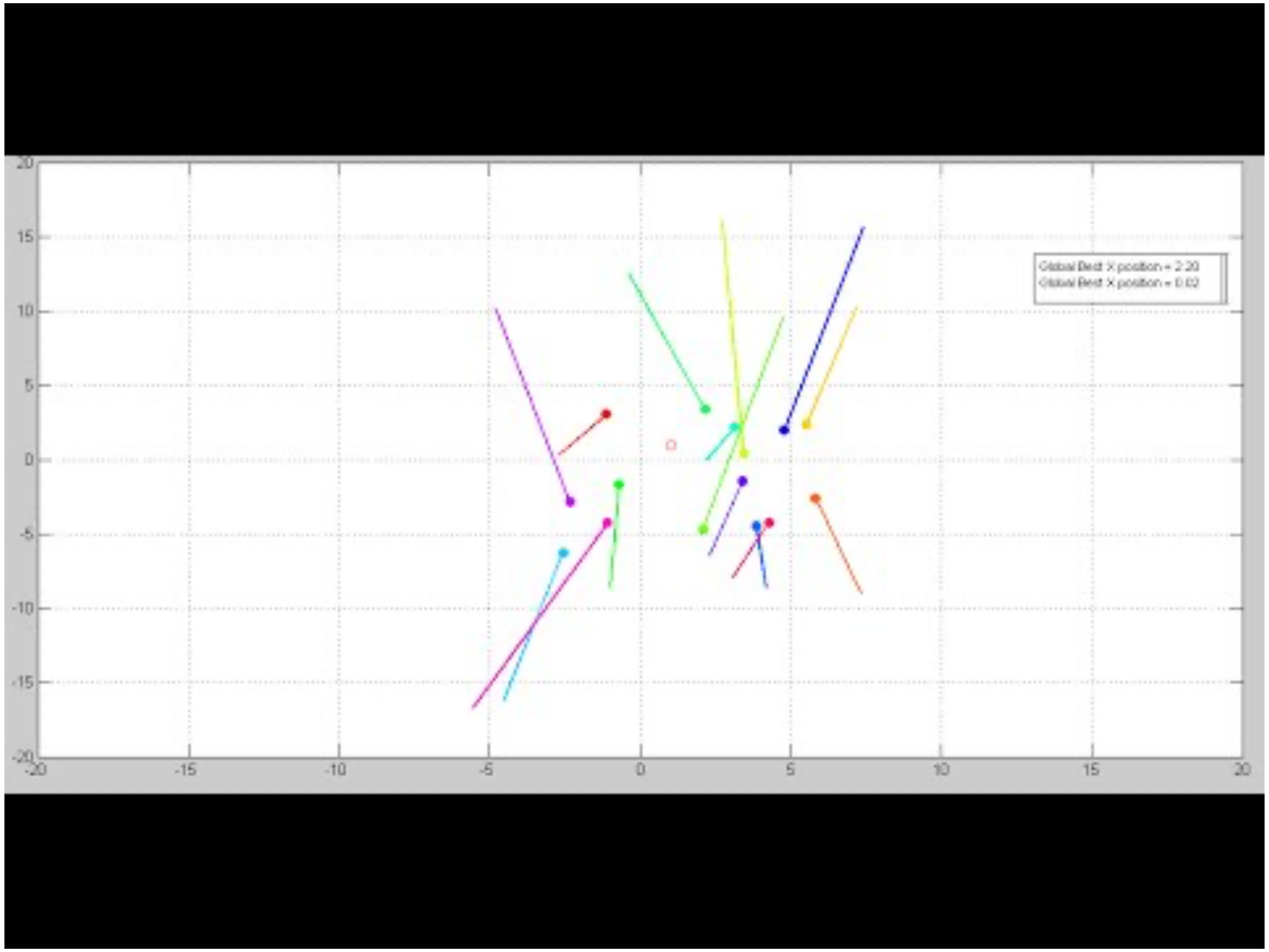
from *Particle Swarm Optimization, Wikipedia*

PSO Example 1



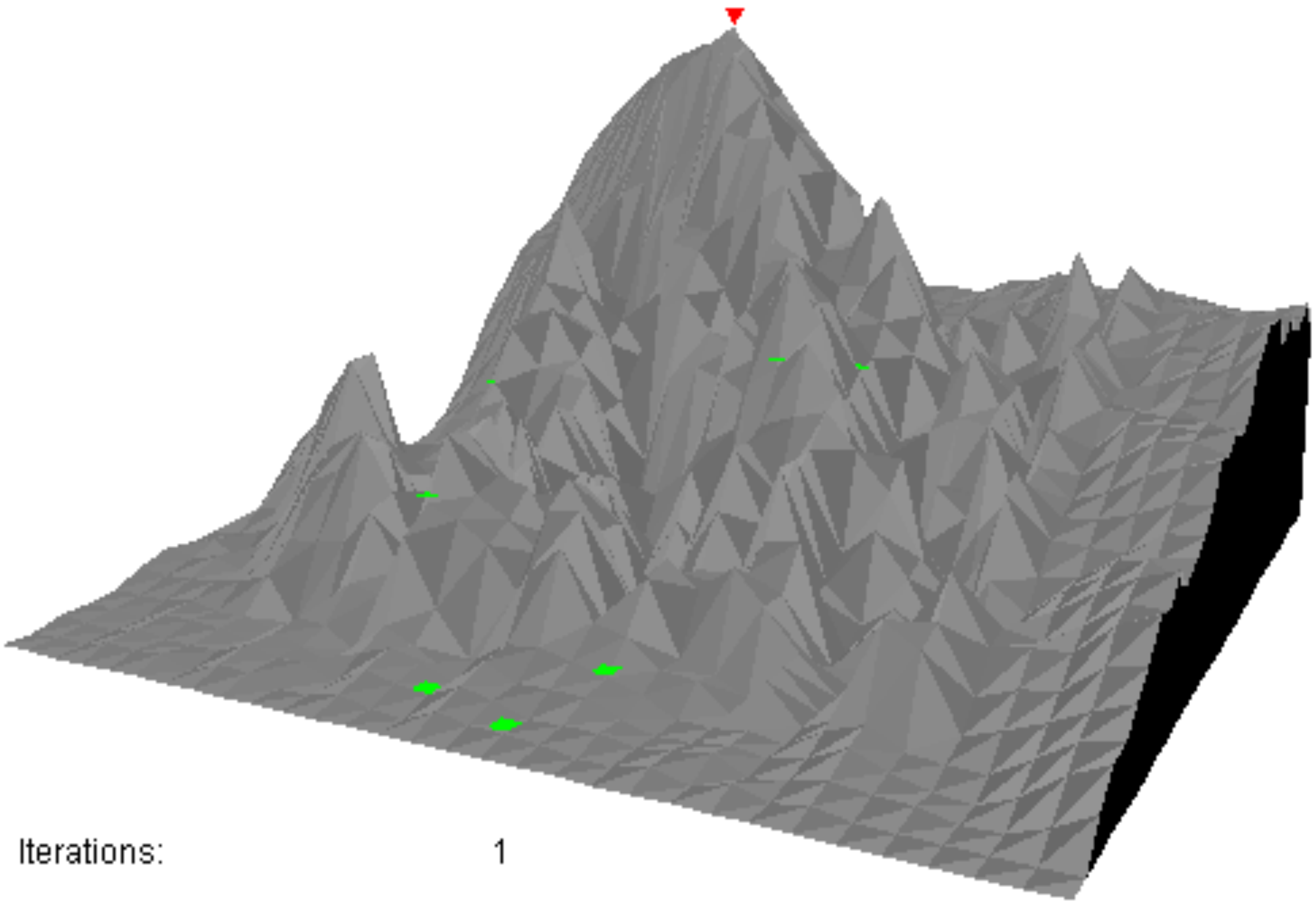
[link to animated gif]()

PSO Example 2

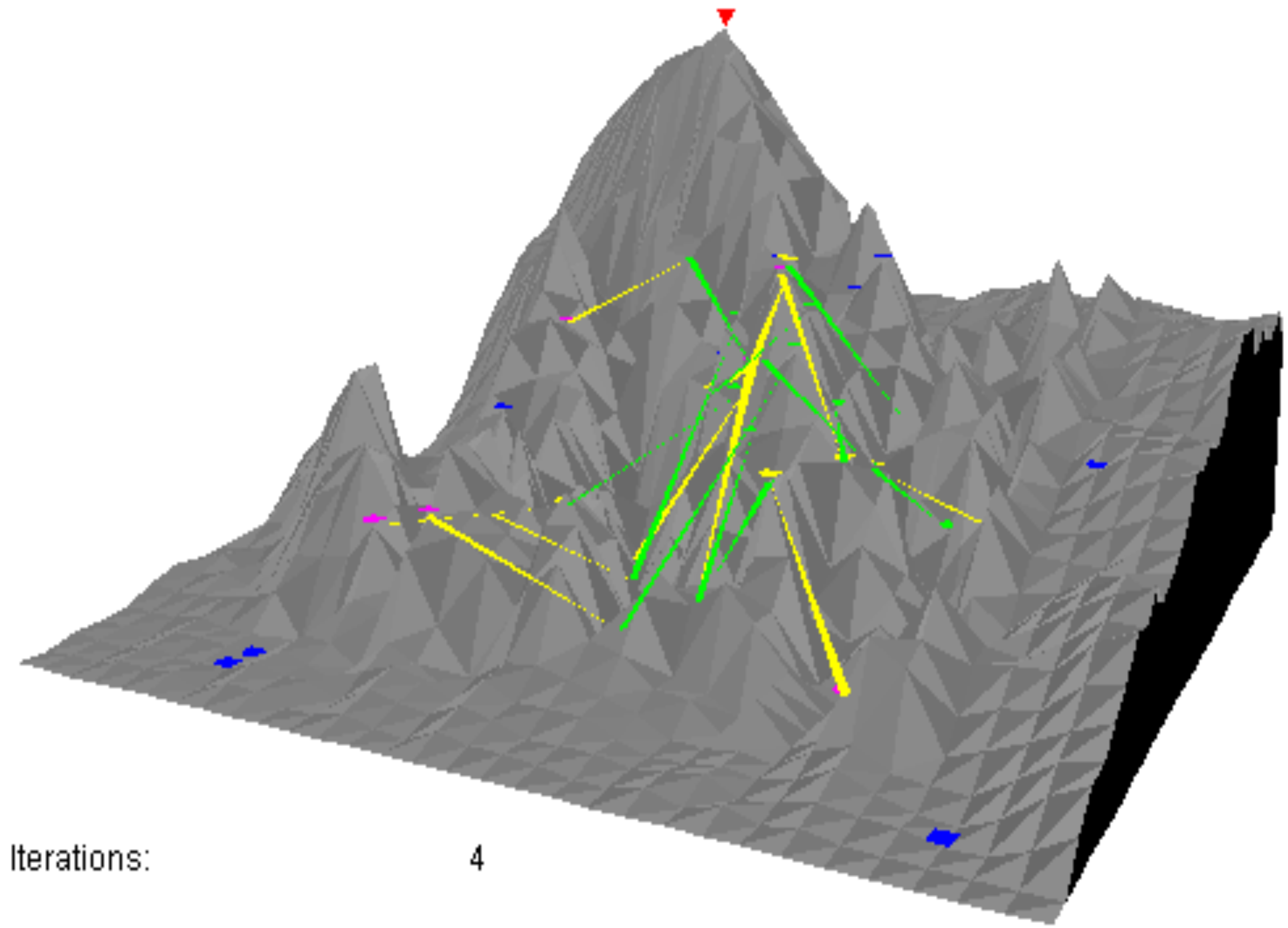


(image link leads to youtube video)

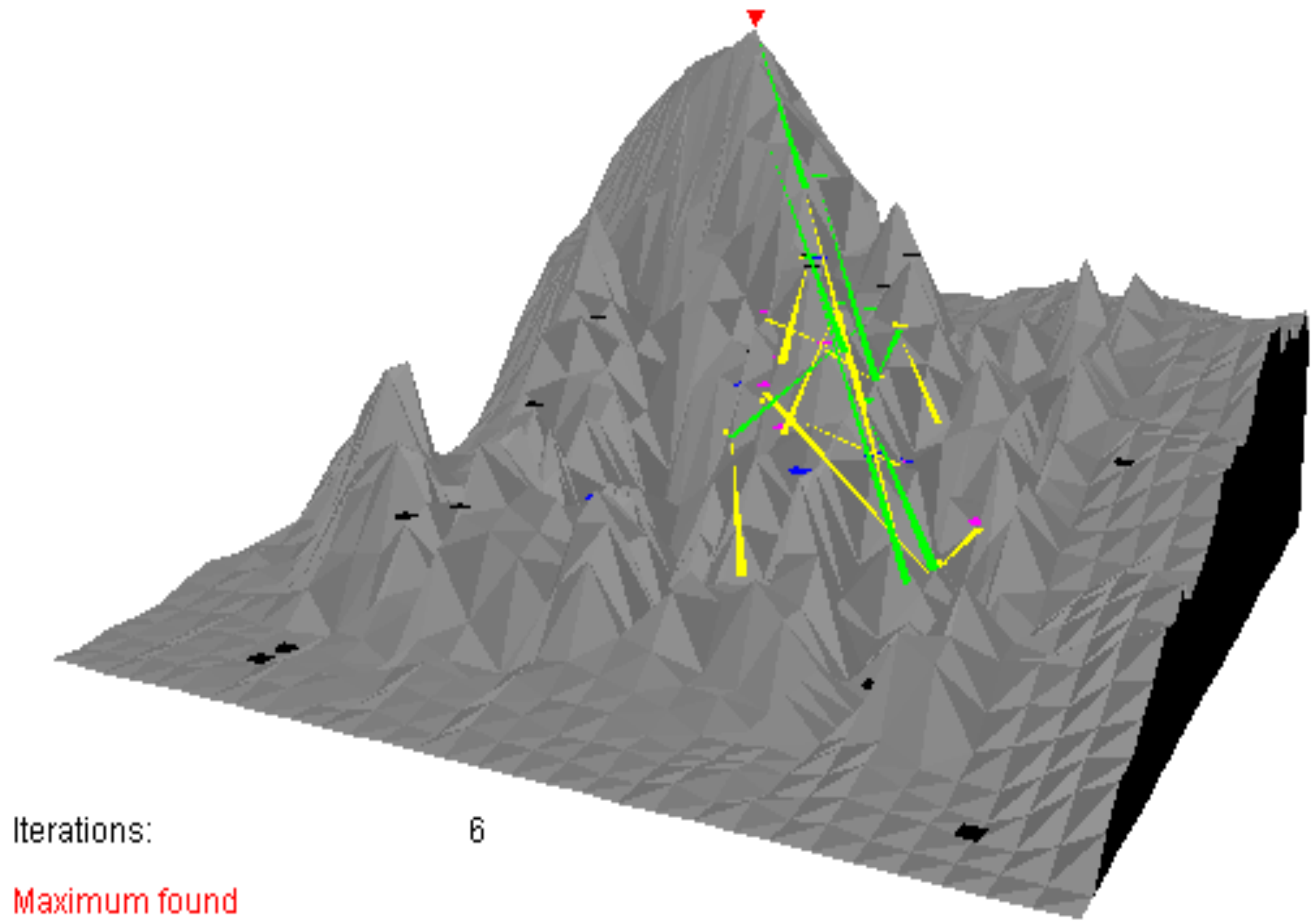
PSO Example 3



PSO Example 3



PSO Example 3

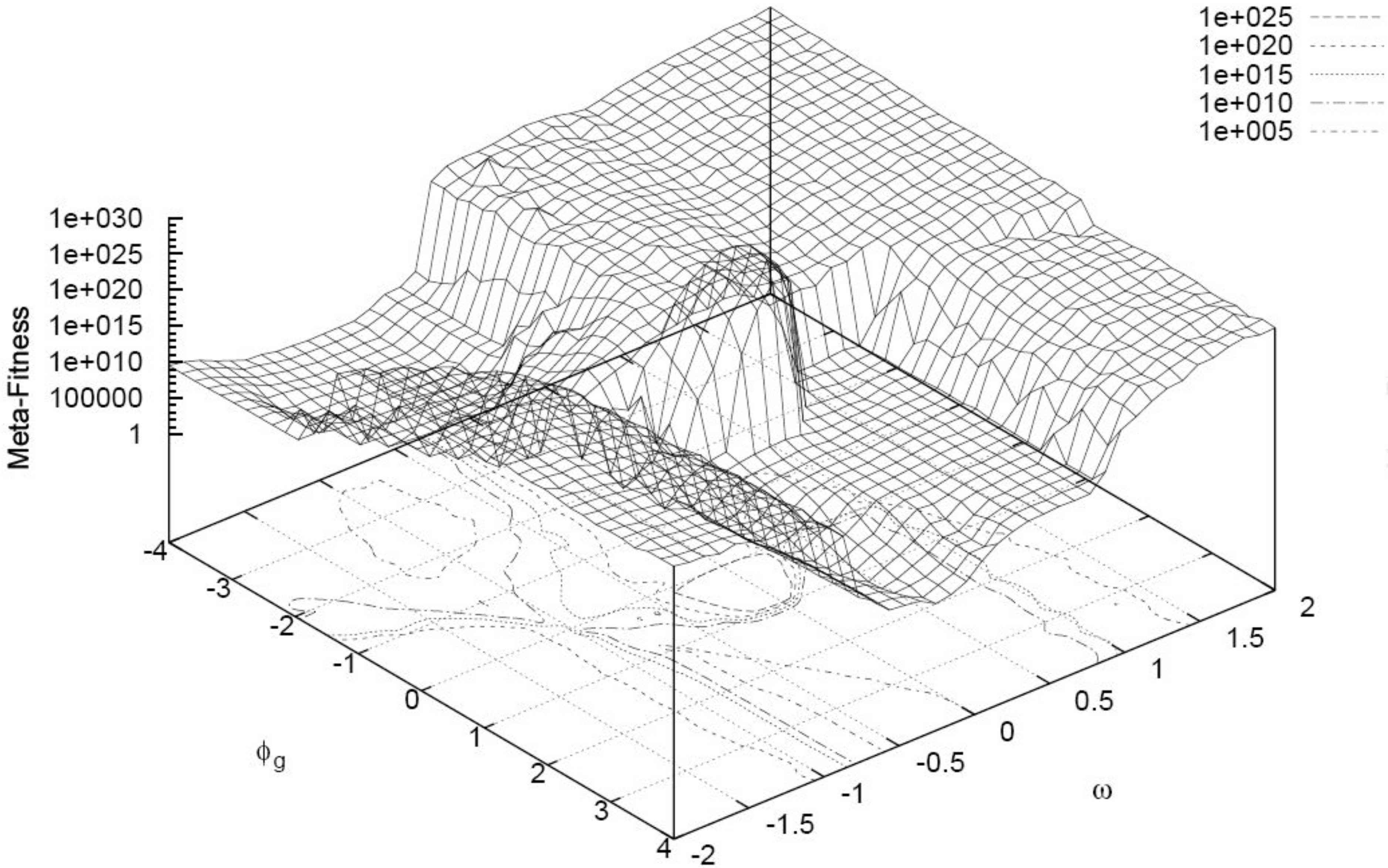


PSO Performance

- relies on selecting several parameters correctly:
- constriction factor
    - used to control the convergence properties of a PSO
- inertia weight
    - how much of the velocity should be retained from previous steps
- cognitive parameter
    - the individual's 'best' success so far
- social parameter
    - neighbours' 'best' successes so far
- b_{lo} & b_{up}
    - the range of possible velocities along any dimension

PSO Performance

- aggregate performance of a simple PSO when two parameters are varied



PSO: Why do they work?

- no clear consensus
- traditional explanation:
  - the swarm behaviour varies between exploratory and exploitative behaviour:
    - exploratory search of a broader region of the search-space
    - exploitative locally oriented search to get closer to an optimum
    - so the PSO algorithm and its parameters must be chosen to balance between these behaviours
      - to ensure a good rate of convergence to the global optimum

PSO: Why do they work?

- alternative explanation:
  - the behaviour of a PSO swarm is not well understood!
    - especially for higher-dimensional search-spaces and complex optimization problems
    - pragmatic approach:
      - for a given problem find a PSO algorithm and parameters that cause good performance regardless of how the swarm behaviour can be interpreted
      - this has led to simplified variants of the PSO algorithm

Applications of PSO

- human tumour analysis
- milling optimisation
- ingredient mix optimisation
- pressure vessel design
  - (a container of compressed air, with many constraints)
- where we want to find the global maxima of a continuous, discrete, or mixed search space, that has multiple local maxima
  - sounds familiar?

PSO are not EA!

- the designs of PSO (and ACO) algorithms are influenced by Evolutionary Algorithms
  - such as the methods of evaluating good solutions
- but there are key differences, such as:
  - the concept of fitness-based selection is not considered in PSO
    - so EA evolve populations over several generations, selecting new individuals based on fitness
    - whereas PSO use a population with a single generation, where those individuals react to their own and their neighbours' successes to move towards a 'better' solution
  - there is no concept of recombination in PSO

Reading & References

- slides based on and adapted from:
  - *Swarm Intelligence* (Corne et al)
  - *Swarm Intelligence* (Mohitz et al)
- recommended reading:
  - "*Swarm Intelligence*", Corne et al, Handbook of Natural Computing, pp 1599-1622, Springer
    - Swarm Intelligence module in Brightspace
- go and play with Boids!
  - javascript (online sim): http://www.harmendeweerd.nl/boids/
  - java code (github): https://github.com/tofti/javafx-boids