Swarm Intelligence and Ant Colony Optimisation

EXTRA READING:

Swarm Intelligence by Eberhart et al, Morgan Kaufmann.

Swarm Intelligence, From Natural to Artificial Systems by Bonabeau, Dorigo, Theraulaz, Oxford University Press.

Papers:

A Simplified Recombinant PSO

Ant colonies for the traveling salesman problem

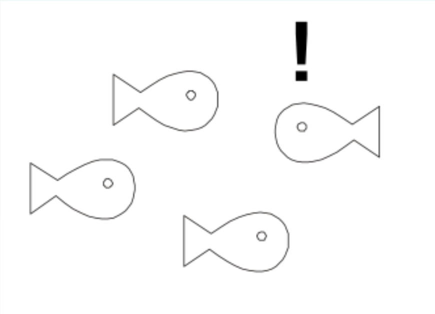
- Swarms of insects, flocks of birds, schools of fish, and herds of wildebeest all have something in common.

- They all move in groups, and the behaviour of the groups is special.
- Somehow, the individuals in the group seem to act in unison.
- They turn together, they flow around obstacles, they move as one.

- Their coordination is so good that it seems as though some centralised controller dictates all movement.
- But this is an illusion. The “cleverness” of their movement is an emergent property of simple rules followed by every individual in the group.
- In other words, the coordination happens as a side effect of *local control* by each individual, not as a result of global control of the whole group.

Flocking Rules of Movement

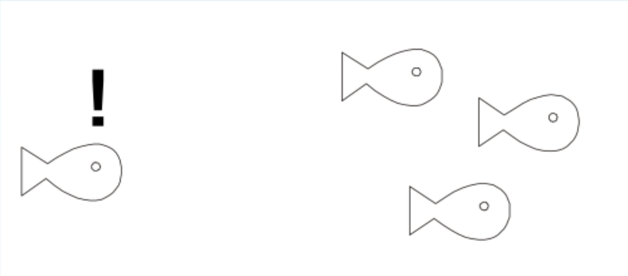
- It does not take many rules to cause realistic flocking or swarming behaviour.
- In 1987 Reynolds developed one of the first simulations of a flock by using just three rules of movement for each individual in the flock:
- **Rule 1 Collision avoidance**



- Avoid hitting any of your companions.

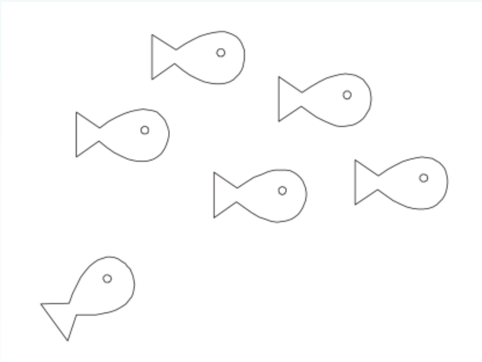
Flocking Rules of Movement

- **Rule 2 Velocity matching**



- Move at the same speed as your companions.

- **Rule 3 Attraction to group centre**



- Try to move towards the centre of the group.

Flocking Rules of Movement

- When implemented in a computer, the result was a very realistic flocking behaviour.

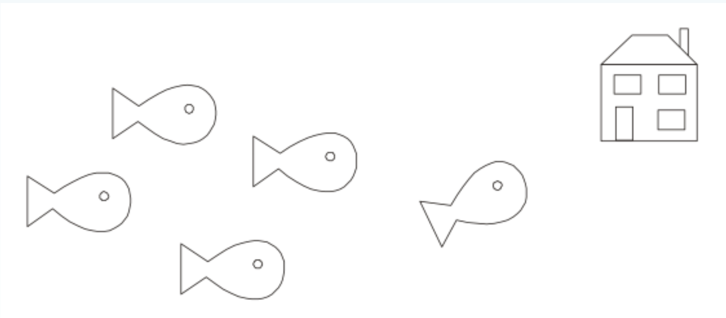
- Reynolds went on to create special effects for movies such as *The Lion King*. (He was responsible for the very realistic movement of the wildebeest in the “stampede sequence”.)

PSO Rules of Movement

- This is not the only example of swarm intelligence. More recently, Kennedy and Eberhart created their version, which they called *particle swarm optimisation.*

- Again, the movement of individuals (now called *particles*) happens according to a set of local rules applied to each particle. These were almost the same as Reynold’s rules except for addition of new rule:

- **Attraction to a ‘roost’ or ‘target’**



- Try to move towards a specific location in space.

PSO

- The reason why they added this rule was because they had realised that a flock or swarm might be capable of solving problems.

- If you recall, an evolutionary algorithm can be regarded as a search algorithm:

- Every member of the population has a specific location in the search space, and every location defines a corresponding solution to a problem.
- We select fitter members of the population (those occupying better positions in the search space) to be parents.
- The children resemble their parents, and so occupy similar, good positions in the space.
- Over a few generations, evolution makes populations converge onto good areas of the search space by searching in parallel.

- Kennedy and Eberhart realised that a swarm might do the same thing.
- If the space that a swarm flew about in, represented a problem space, then every location in the space would define a solution to the problem, just like before.
- So a swarm with many particles would be sampling many possible solutions at once, like a population of an evolutionary algorithm.
- Instead of evolving using crossover and mutation to find new locations, the particles would swarm around in the space, following their rules of movement.

- They added their new rule (attraction to a target) to encourage the swarm to find the solution to a problem.
- By being attracted to a target, the swarm was being attracted to better solutions.
- Using a fitness function to measure how good each solution is, the swarm could “smell” good places to be in the space.
- And because particles like to follow each other, and stay close to each other, if one finds a good solution, the others quickly follow, swarming and exploring all of the nearby solutions.
- Let’s look at the particle swarm rules in more detail.

1 Particle Swarm dynamics

- $\Delta v(t) \quad = \quad F\,(\,x(t-1), \Delta v(t-1), p_b, p_g\,)$

- The particle acceleration can be a function F of:
- the particle position x(t)
- the previous acceleration value $\Delta v(t-1)$
- the particles' best position (p_b)
- the local neighbourhood's best position (p_g), where 'best' is evaluated with respect to some cost function
- and anything else you think might be useful…

2 Particle Swarm velocity update

- v(t) = v(t-1) + Δv(t-1)

- Velocity at time t is velocity at time t-1 plus the acceleration value.

3 Particle Swarm max velocity

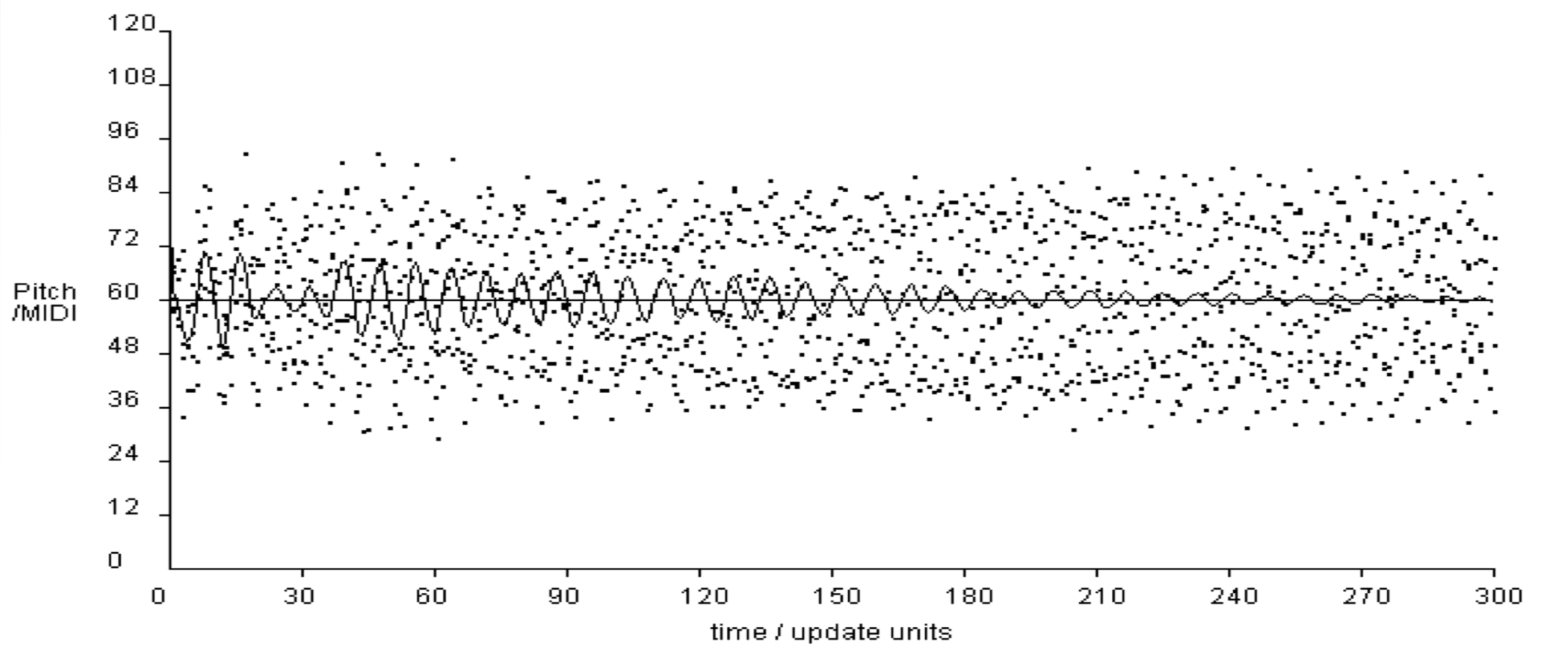
- $v(t + \varepsilon) = v(t) + \theta(\ |(v(t) / v_{max})|\ - 1)\ (v_{max} - v(t))$
- This provides a nonlinear damping force which is applied instantaneously and has the effect of limiting the velocity magnitude to the cut-off v_{max}.
- θ is the step function defined by $\theta(a) = 0$ for $a < 0$, $\theta(a) = 1$ otherwise.
- It is used to control unbounded oscillations of the swarm around the target solution. (Other functions may be used.)

4 Particle Swarm position update

- $x(t) = x(t-1) + v(t)$.
- Particle position at time t is position at time t-1 plus the velocity value.

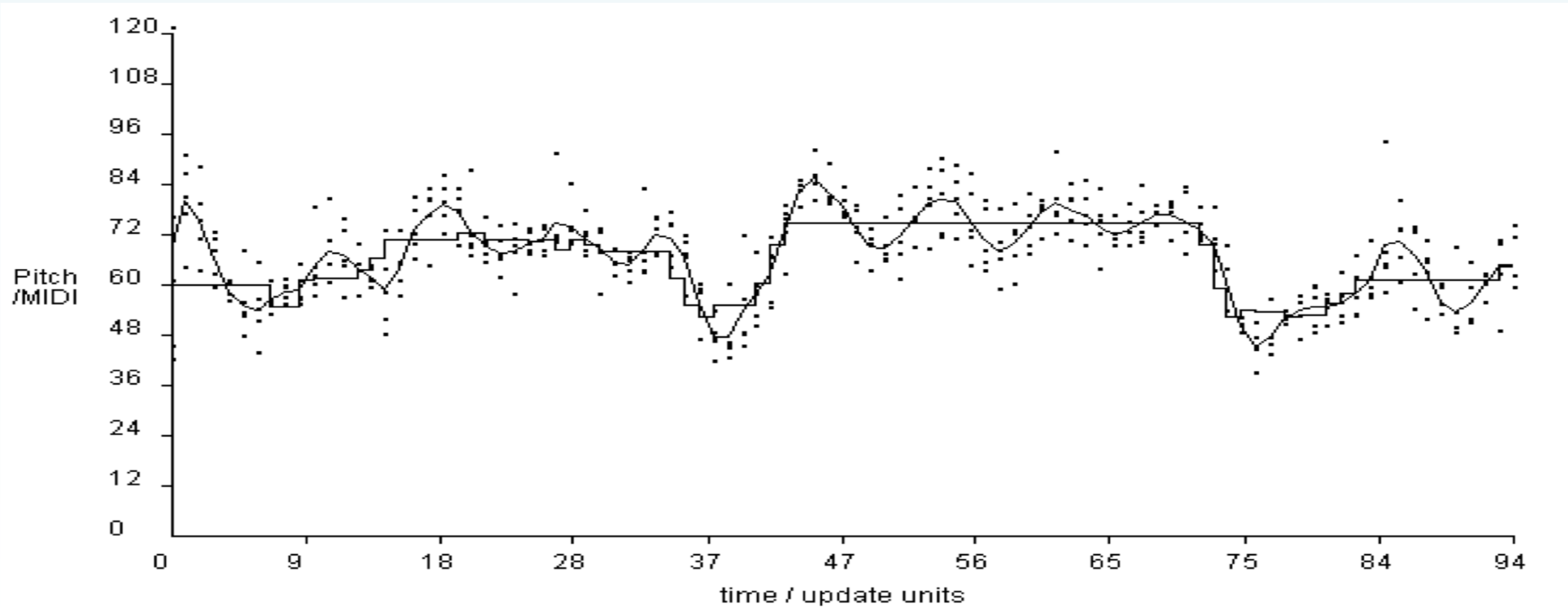
PSO RULES OF MOVEMENT

- These simple rules cause the particles in the swarm to find good solutions quickly, swarm around them, and settle on the best.
- To see this, look at the plot of a swarm finding a stationary target (the straight line at 60).
- The centre of the swarm over time is plotted, and shows how the particles oscillate around the target, but soon converge onto it.



PSO RULES OF MOVEMENT

- Additionally, swarms are constantly moving and do not converge genetically like individuals in an evolutionary algorithm.
-
- So, should the target (i.e., problem) be continuously changing, the swarm is able to constantly move and find a good solution in real time.
-
- The following plot shows this happening: note the way the swarm centre is always close to the moving target.



- Today the standard PSO inertia weighted formalism usually resembles:

$$\mathrm{IW}: v_{id}^{t+1} = w v_{id}^{t} + \frac{\phi}{2} u_1 (p_{id} - x_{id}^{t}) + \frac{\phi}{2} u_2 (p_{nd} - x_{id}^{t})$$

- where *d* labels components of the position and velocity vectors, *d* = 1, 2, …, D,
- $\vec{p}_i$ is the personal best position achieved by *i*,
- $\vec{p}_n$ is the best position of informers in *i* 's social neighborhood, and
- $u_{1,2} \sim U(0,1)$.

- After velocity update, the particle position is adjusted:

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

PSO RULES OF MOVEMENT

- Peña et al. introduced a recombinant version of PSO by replacing either the personal best or the neighbourhood best position by the recombinant position.
- The former provides improved performance and has the more interesting social aspect. A recombinant position vector $\vec{r}$ is defined by

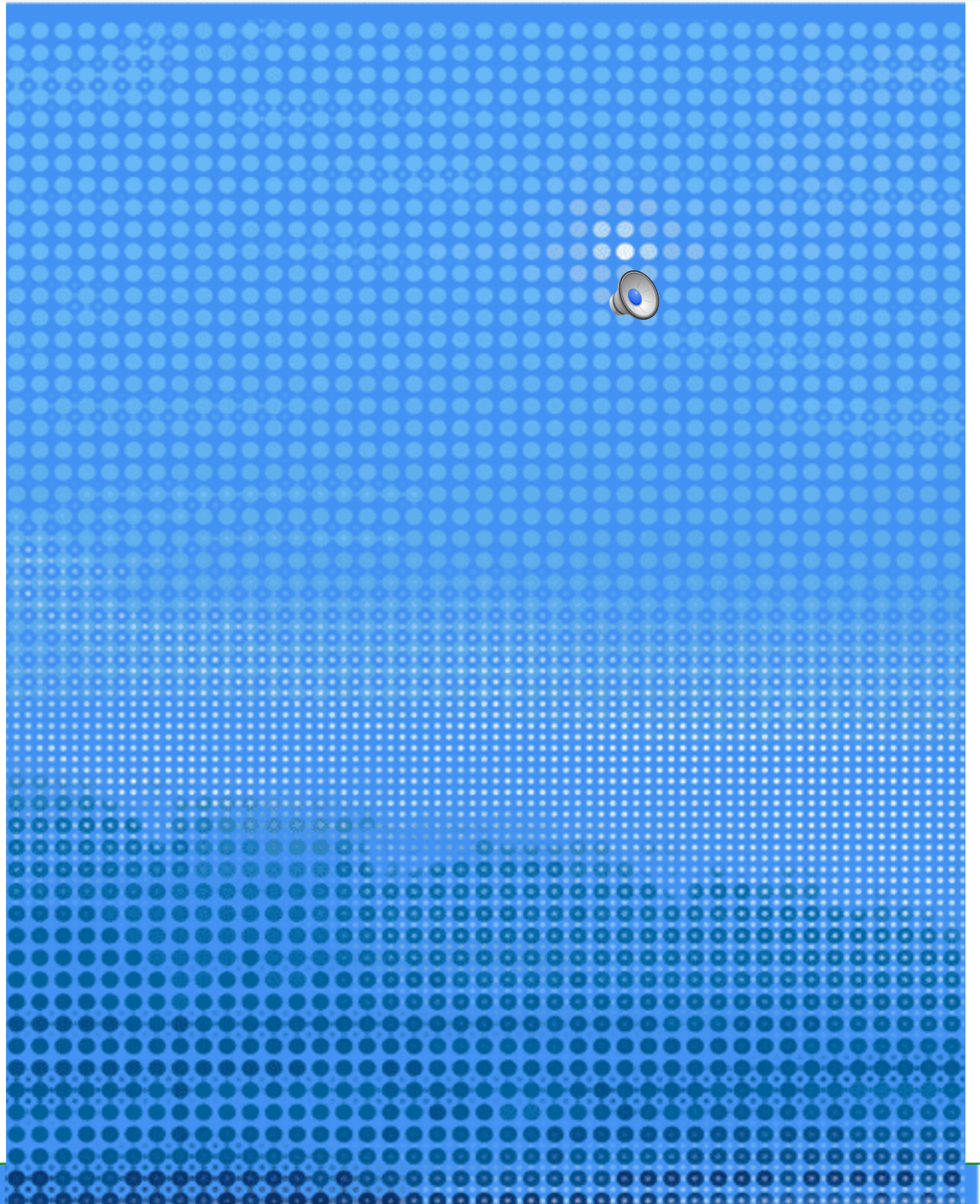
$$r_{id} = \eta_d p_{ld} + (1 - \eta_d) p_{rd}$$

- where $\eta_d = U\{0,1\}$ and
- $\vec{p}_{l,r}$ are immediate left and right neighbors of *i* in a ring topology.
- While separate random numbers η_d are used for separate dimensions *d*, a single value is generated for each single dimension and used for both occurrences of η_d in that dimension. This places $\vec{r}_i$ at a corner of the smallest *D*-dimensional box which has p_l and p_r at its corners.

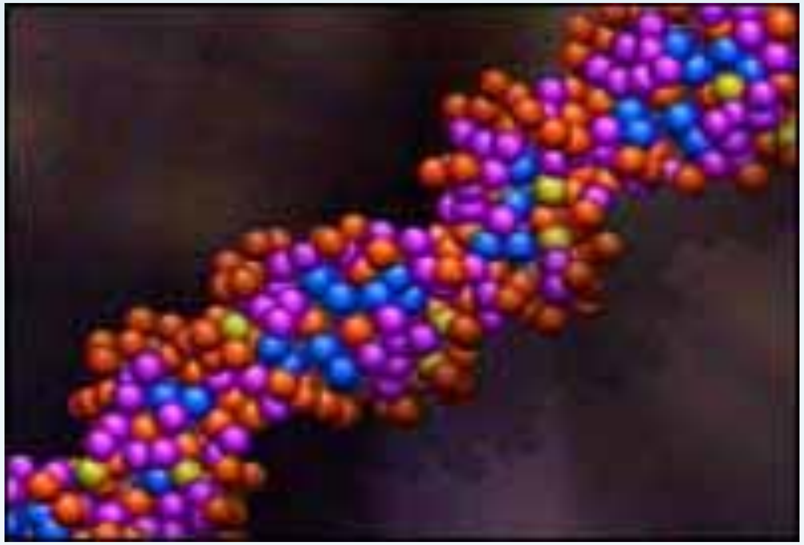
PSO RULES OF MOVEMENT

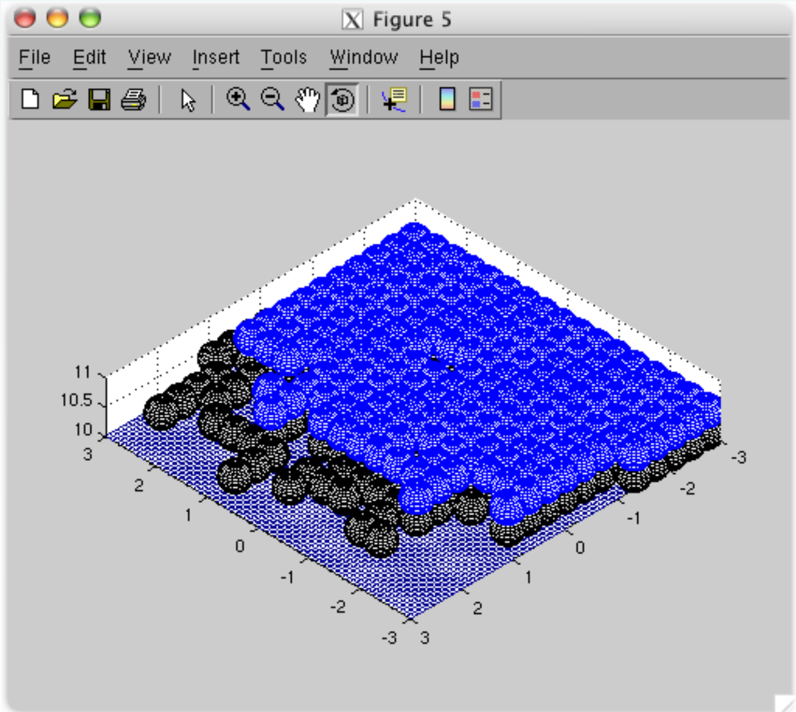
- The plots shown previously are from an application tried out at UCL.
- The swarm responded to real audio input (such as a singer or saxophone).
- The input was treated as a target, and the swarm moved in “music space” towards that target, with the location of every point defining a musical note.
- The result was a program that could improvise music with a musician in real time.

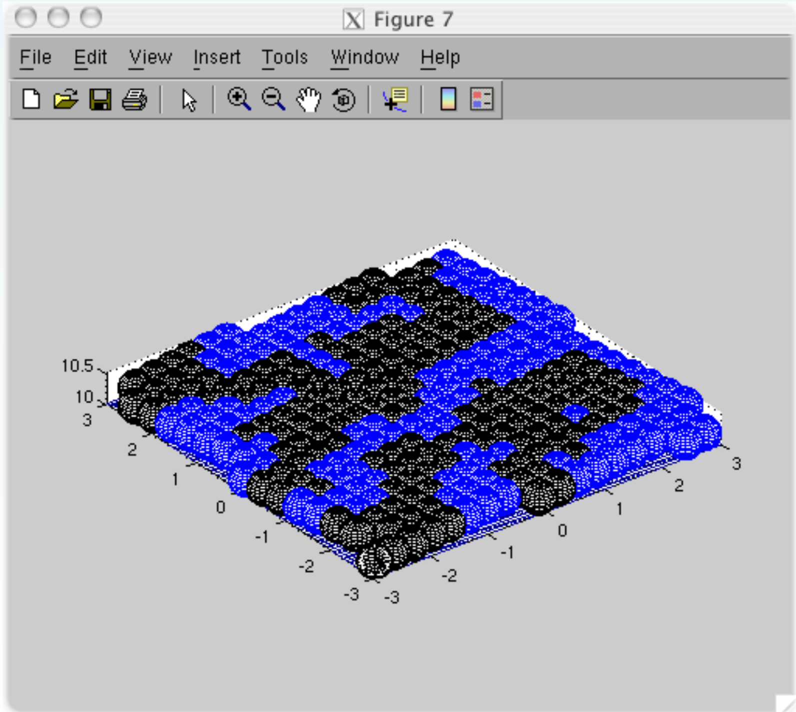
Tim Blackwell's Swarm Music



Boonserm Kaewkamnerdpong's swarming nanotech







Navneet Bhalla's physical evolved self-assembly

Self Organisation

- Swarm intelligence, and indeed many of the phenomena we see in “intelligent” insect behaviour, arises because of the emergent effects of local rules.
- Termites build extraordinary structures, ants forage and have complex societies, bees make complex decisions about which food sources to use.
- It all seems as though there is one big brain somewhere, controlling everything.
- These phenomena occur because of *self-organisation*.

Self Organisation

- Experts on insect behaviour have borrowed ideas from physics and chemistry to explain insect intelligence.

- According to these theories, complexity can arise spontaneously, if certain conditions are met. These are:

- **Interaction**
- **Positive feedback**
- **Negative feedback**
- **Amplification of fluctuations**

Self Organisation

- To see how this explains swarm intelligence, let’s go through each in turn.
- Remember the simple movement rules of our swarm?
- They force **interaction** between the particles in the swarm.
- If one gets too close to another, they will both try to move apart.
- If one “smells” a good place to go, others will follow.
- The movement of each particle affects the movement of the others.

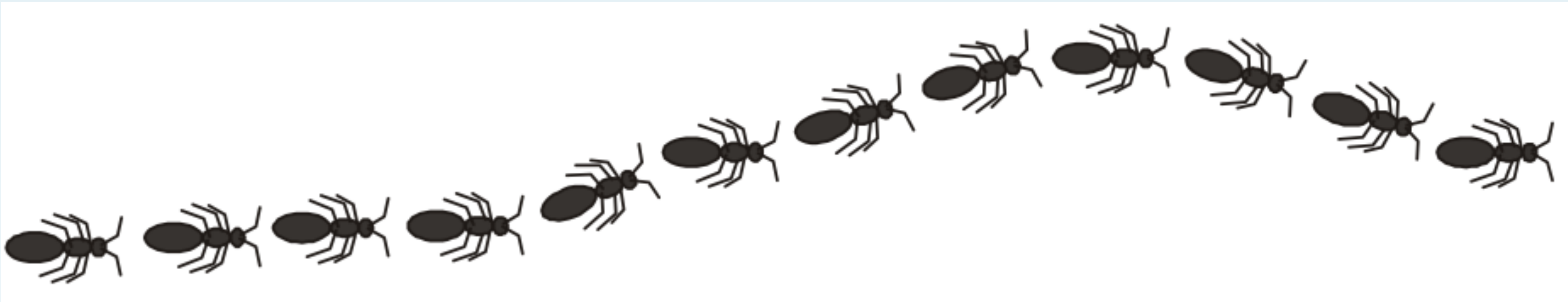
Self Organisation

- Suppose a particle randomly happens to fly through the target – a place it *really* wants to be.
- It will stay in that region and its companions will soon follow.
- Why?
- Because particles are attracted to each other, and so any nearby will be attracted to the first particle and the target as well, making a double attraction.
- So the lucky find or **fluctuation** will be **amplified** as the whole swarm soon moves over the target.

Self Organisation

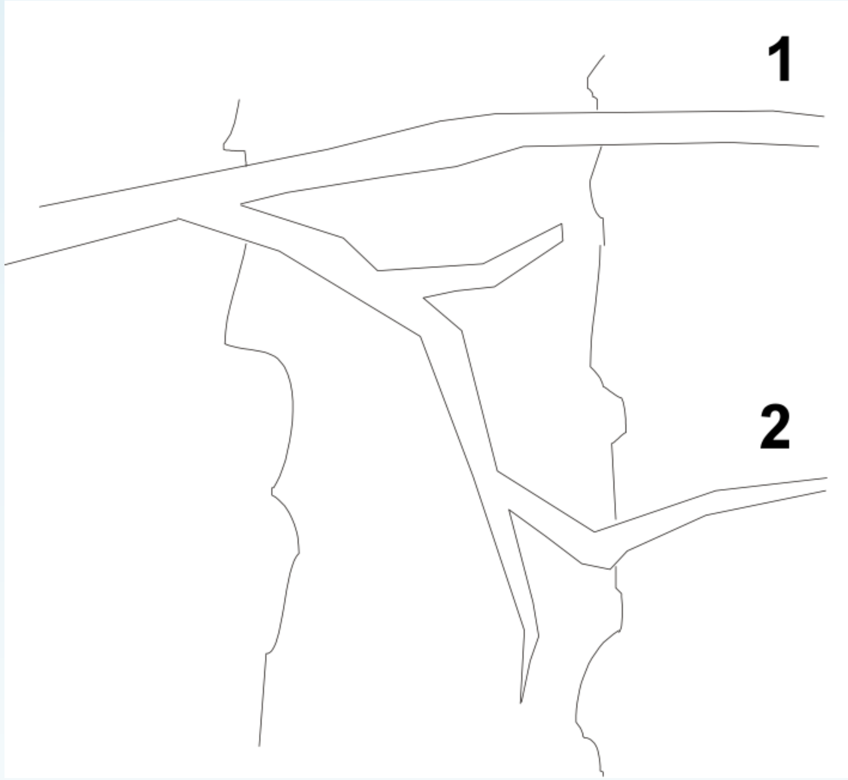
- The **positive feedback** happens in much the same way.
- The more particles there are in one place, the more “pull” will be exerted on other particles anywhere else.
- Any finally, **negative feedback** is caused by the ‘max velocity’ weighting.
- Also, if the velocity starts to get so high that a particle might fly off and never be seen again, the max velocity weighting will pull it back.
- This provides a brake for the positive feedback.

ANT COLONY OPTIMISATION

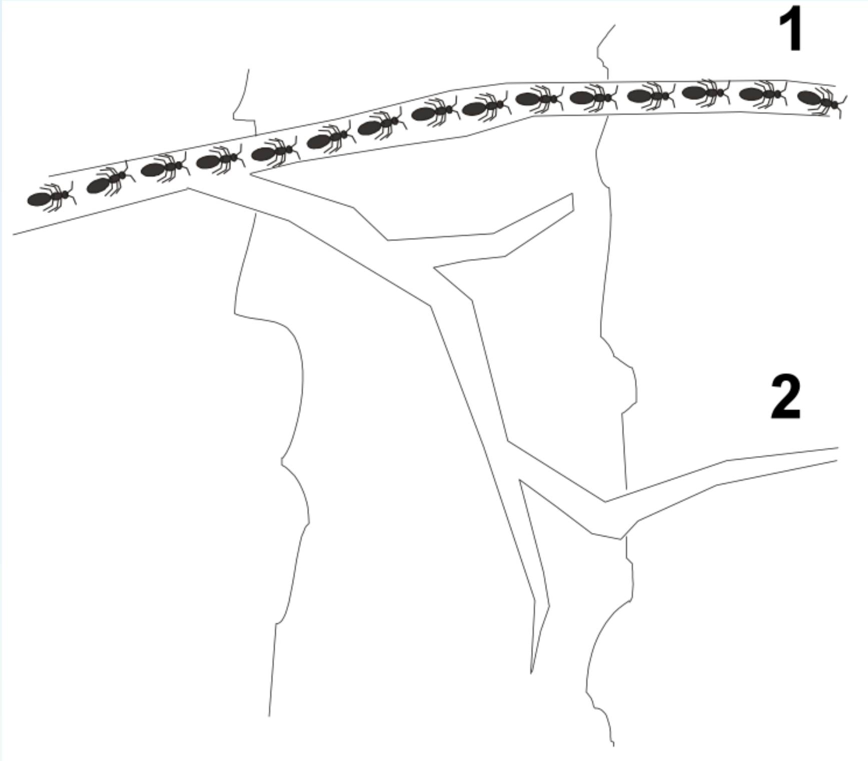


- Ants may not be very clever individually, but ant colonies can be
- An ant colony is capable of searching, making plans, and optimising routes to food.
- Ant colonies are so good at finding the shortest path from one location to another, that we have developed an algorithm based on their behaviour.
- Its name is *Ant Colony Optimisation.*

ANT COLONY OPTIMISATION



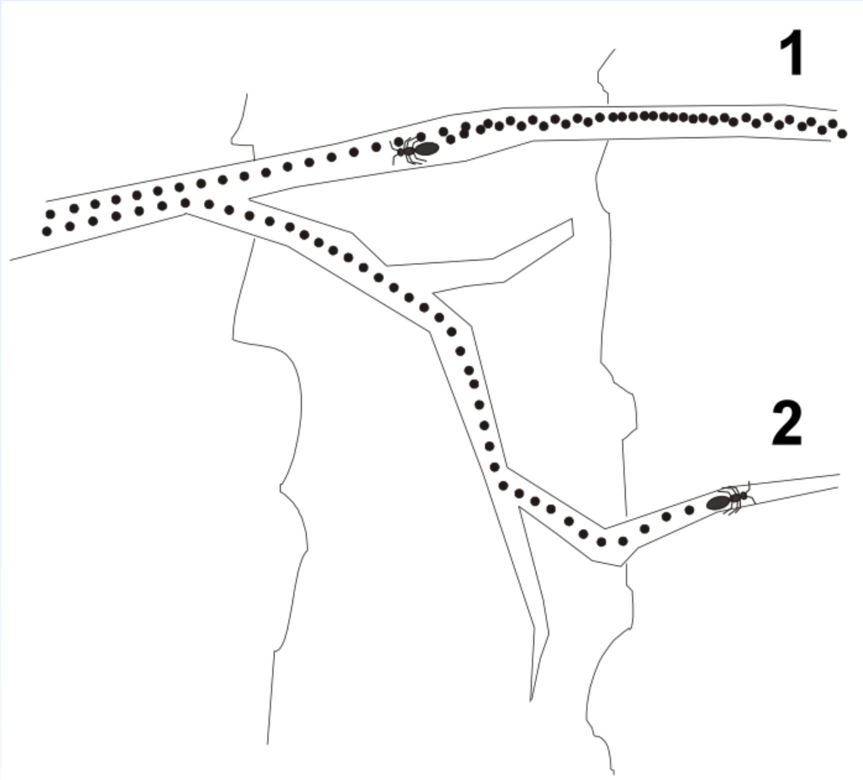
- Imagine the situation above.
- A river or hole separates the nest from a food source.
- There are two ways to cross: a short, direct path, or a longer, less efficient path.
- Thousands of ants need to make this journey every day. If even a few choose the longer path, they waste time and energy.
- So, what do the ants do?



- They take the shorter path.
- But how do they choose?
- A single ant is not intelligent enough to make this choice. How can a colony be cleverer?
- The answer has to do with pheromones, or smelly trails.

ANT COLONY OPTIMISATION

- Every ant leaves behind a smelly trail.
- In the time it takes one ant to cross using the longer route, the other ant has almost returned on the shorter route, still laying pheromone as it walks.
- So there is more pheromone on the shorter path than on the longer one. The pheromone attracts other ants, which also lay down pheromone.



- Very quickly, the amount of pheromone is so strong on the shorter path, that all the ants take this route.

- Indirect communication by modifying your environment is called **stigmergy**.
- This is another example of self-organisation. Can you work out why?

ANT COLONY OPTIMISATION

- Ant colonies are good at finding shortest paths.
- This is exactly the ability we need in order to solve Travelling Salesman Problems (TSPs).
- A TSP involves finding the shortest tour of cities, where every city is visited.
- It is the same class of problem as routing in networks.
- For three or four cities, this problem is easy. But for fifteen or twenty, or more, the problem is very difficult.
- Marco Dorigo created a new algorithm based on the behaviour of ants to solve TSPs.

- The algorithm is simple: first the ants explore, by choosing different tours of the cities. The better tours have pheromone levels increased, and the process repeats.

- An artificial ant *k* in city *r* chooses the city *s* to move to, amongst those that do not belong to its working memory M_k by applying the following probabilistic formula:

$$s = \begin{cases} \arg\max_{u \notin M_k} \left\{ [\tau(r,u)] \cdot [\eta(r,u)]^{\beta} \right\} & \text{if } q \leq q_0 \\ S & \text{otherwise} \end{cases}$$

where:

$\tau(r,u)$ is the amount of pheromone trail on edge (r,u)

$\eta(r,u)$ is the heuristic function: $\frac{1}{dist(r,u)}$

β is a constant defining relative importance of pheromone trail and closeness

q is a random number between 0 and 1

q_0 is a threshold constant between 0 and 1

- S is a random variable selected according to the following probability distribution, which favours shorter edges that have a higher level of pheromone trail:

$$p_k(r,s) = \begin{cases} \dfrac{[\tau(r,s)] \cdot [\eta(r,s)]^{\beta}}{\sum_{u \notin M_k} [\tau(r,s)] \cdot [\eta(r,s)]^{\beta}} & \text{if } s \notin M_k \\ 0 & \text{otherwise} \end{cases}$$

where:

$p_k(r,s)$ is the probability with which ant k chooses to move from city r to city s

- Pheromone trails on the edges between cities are changed *locally* and *globally.*

- Global updating rewards edges belonging to shorter tours of cities.

- Once artificial ants have all completed their tours, the ant that has travelled the shortest distance deposits additional pheromone on each edge it visited

- The amount of pheromone $\Delta\varphi(r,s)$ deposited on each visited edge (r,s) by the best ant is inversely proportional to the length of the tour: the shorter the tour, the greater the amount of pheromone deposited on the edges.

- This manner of depositing pheromone is intended to emulate the actions of many ants as they explore and increase the levels of pheromone on shorter paths.

- The global trail updating formula is:

$$\varphi(r,s) \leftarrow (1-\alpha)\cdot \varphi(r,s) + \alpha \cdot \Delta\varphi(r,s)$$

where

$\Delta\varphi(r,s)$ is $\dfrac{1}{\textit{shortest tour}}$

α is a constant defining the relative importance of the shortest tour distance and the existing pheromone level.

- Note that global trail updating is very similar to a reinforcement learning scheme in which better solutions get a higher reinforcement.

- In addition to global trail updating, local trail updating is used.

- To avoid a very strong edge being chosen by all of the ants: every time an edge is chosen by an ant, its pheromone is updated by the local trail updating formula:

$$\tau(r,s) \leftarrow (1-\alpha)\cdot\tau(r,s) + \alpha\cdot\tau_0$$

  where τ_0 is a system parameter

This is also intended to model trail evaporation.

- So when an ant chooses its tour, it will either:
- exploit the experience accumulated by the ant colony in the form of pheromone trails (with probability q_0), or
- explore randomly with a bias towards short and high pheromone trail edges not already visited
- The result is an algorithm capable of searching in parallel and finding solutions to TSP problems very successfully.
- The inventors showed that this algorithm can find perfect solutions to 100-city TSPs, where algorithms such as evolutionary programming, genetic algorithms and simulated annealing struggle.

Questions?

Kumar, S. and Bentley, P. J. (Contributing Eds.) (2003) ***On Growth, Form and Computers***. Academic Press, London.

Peter J. Bentley (2002). ***Digital Biology. How nature is transforming our technology and our lives.*** Simon & Schuster (USA Hardback). ISBN: 0743204476

Bentley, P. J. and Corne, D. W. (Contributing Eds.) (2001) ***Creative Evolutionary Systems.*** Morgan Kaufmann Publishers Inc., San Francisco, CA.

Bentley, P. J. (Contributing Editor) (1999). ***Evolutionary Design by Computers.*** Morgan Kaufmann Publishers Inc., San Francisco, CA.