Genetic Algorithms and Genetic Programming

Lecture 14: (13/11/09)

Particle Swarm Optimization

School of informatics

Michael Herrmann

michael.herrmann@ed.ac.uk, phone: 0131 6 517177, Informatics Forum 1.42

Overview

I. GA (1-7)

II. GP (8-10)

III. ACO (11-13): Ant colony optimization

IV. PSO (14-15): Particle swarm optimization and differential evolution

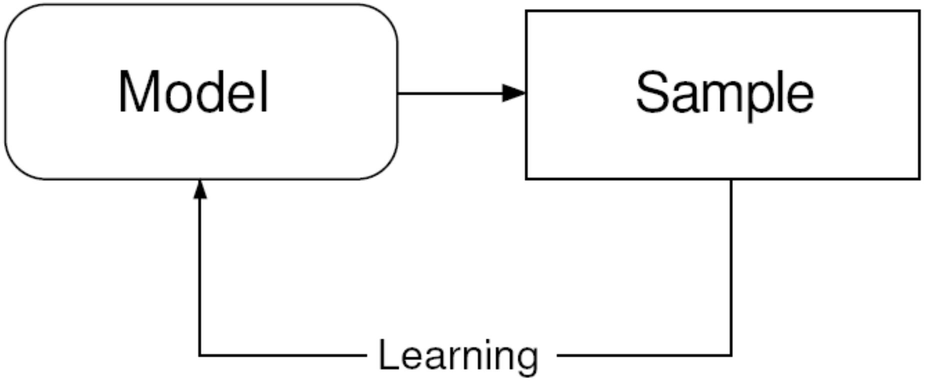
V. NC (16): Overview on DNA computing, Membrane computing, Molecular computing, Amorphous computing, Organic computing, ….

VI. Wrapping up: Metaheuristic search (17)

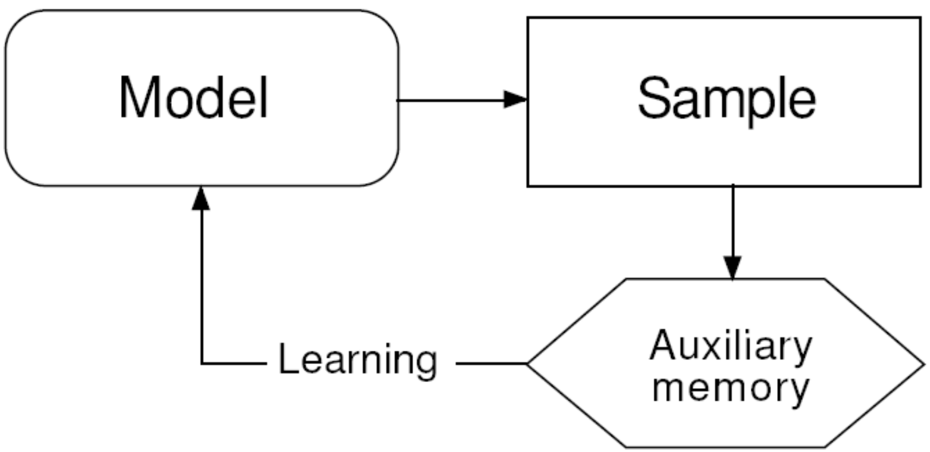
Not included:

artificial neural networks, quantum computing, cellular automata, artificial immune systems

Relation to other algorithms: Model-Based Search



Scheme of the MBS approach



MBS approach with memory

E.g. in ACO:

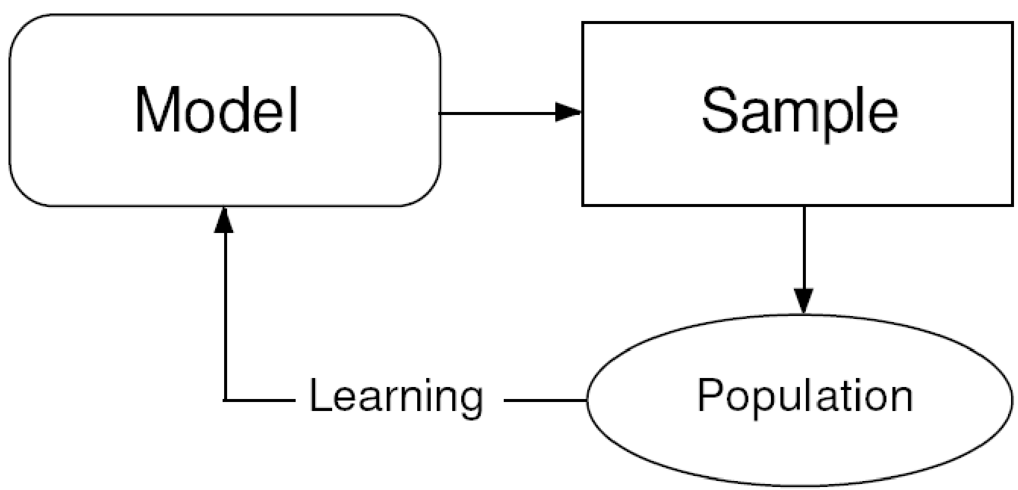
- Model: pheromone matrix
- Sample: ants following pheromone traces
- Learning: pheromone update

- Auxilary memory: best-so-far solution

Zlochrin, Birattari, Meuleau, Dorigo: Model-based Search for Combinatorial Optimization: A Critical Survey. Annals of Operations Research 2004.

GA as MBS

- Generate new solutions using the current probabilistic model
- Replace (some of) the old solutions by the new ones.
- Modify the model using the new population.



GA as MBS

compact Genetic Algorithm (cGA) (Harik et al., 1999)

- Probabilistic simulation of a genetic algorithm with tournament selection
- Probabilistic model of the population: individual are generated by biased draws based on a probability vector. E.g. if the vector entry p_i is 0.9 it is likely to have a 1 at position i in this individual's string.
- Tournament selection: Choose two individuals *a* and *b*

$$
\begin{array}{l}
\text{if } a_i \neq b_i \text{ then} \\
\quad \text{if } a_i = 1 \quad \text{then} \quad p_i \leftarrow p_i + 1/n \\
\quad \phantom{\text{if } a_i = 1} \quad \text{else} \quad\; p_i \leftarrow p_i - 1/n
\end{array}
$$

- The model is updated by

$$p_i \leftarrow p_i + \frac{1}{n}(a_i - b_i)$$

GA as MBS

- Bits in the genome were chosen independently. What about schemata?
- Modeling dependencies between string positions e.g.
  - learning a chain distribution as in ACO starting at the first character of the string and setting the next one by a conditional probability
  - by a matrix of pair-wise joint frequencies
  - by a forest of mutually independent dependency trees
- In order to capture the essential idea of GA (building blocks the probabilistic model must be different from the ACO model (i.e. the pheromone matrix + update)

Swarm intelligence

- Collective intelligence: A super-organism emerges from the interaction of individuals
- The super-organism has abilities that are not present in the individuals (‘is more intelligent’)
- “The whole is more than the sum of its parts”
- Mechanisms: Cooperation and competition … and communication
- Examples: Social animals, smart mobs, immune system, neural networks, internet, swarm robotics

Beni, G., Wang, J. Swarm Intelligence in Cellular Robotic Systems, Proc. NATO Adv. Workshop on Robots and Biological Systems, Tuscany, Italy, 26–30/6 (1989)

Swarm intelligence: Application areas

- Biological and social modeling
- Movie effects
- Dynamic optimization
  - routing optimization
  - structure optimization
  - data mining, data clustering
- Organic computing
- Swarm robotics

Swarms in robotics and biology

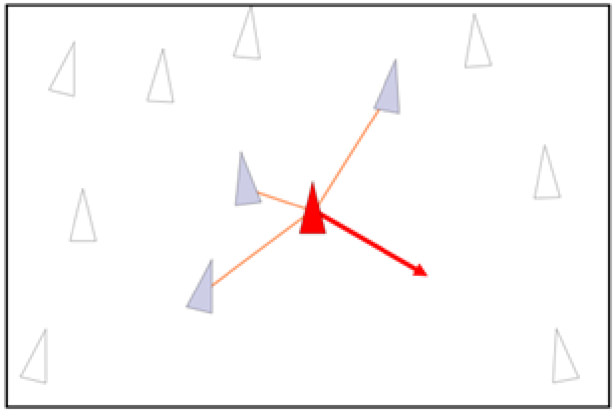
- Robotics/AI
  - Main interest in pattern synthesis
    - Self-organization
    - Self-reproduction
    - Self-healing
    - Self-configuration
  - Construction
- Biology/Sociology
  - Main interest in pattern analysis
    - Recognizing best pattern
    - Optimizing path
    - Minimal conditions
    - not “what”, but “why”
  - Modeling

Dumb parts, properly connected into a swarm, yield smart results.

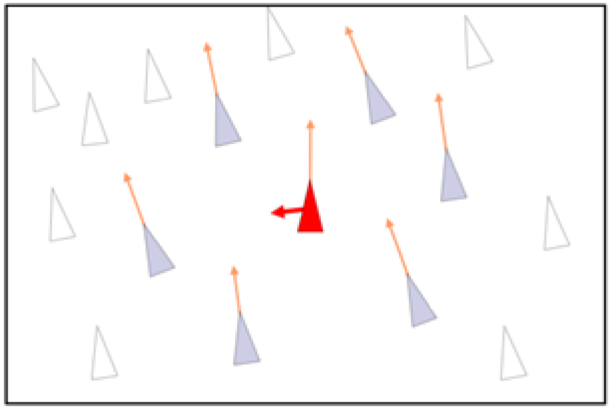
Kevin Kelly

Complex behaviour from simple rules

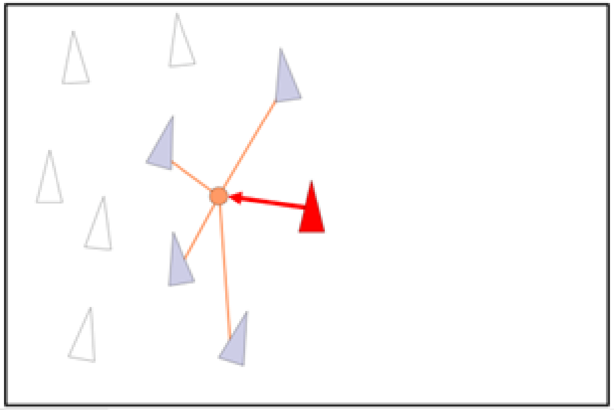
Rule 1: Separation
Avoid Collision with neighboring agents



Rule 2: Alignment
Match the velocity of neighboring agents



Rule 3: Cohesion
Stay near neighboring agents



Towards a computational principle

- **Evaluate** your present position
- **Compare** it to your previous best and neighborhood best
- **Imitate** self and others

Hypothesis: There are two major sources of cognition, namely, own experience and communication from others.

Leon Festinger, 1954/1999, Social Communication and Cognition

Particle Swarm Optimization (PSO)

- Methods for finding an optimal solution to an objective function
- Direct search, i.e. gradient free
- Simple and quasi-identical units
- Asynchronous; decentralized control
- ‘Intermediate’ number of units: ~ 10^{1}-$10^{<<23}$
- Redundancy leads to reliability and adaptation
- PSO is one of the computational algorithms in the field of swarm intelligence (the other is ACO)

J. Kennedy, and R. Eberhart, *Particle swarm optimization*, in Proc. of the IEEE Int. Conf. on Neural Networks, Piscataway, NJ, pp. 1942–1948, 1995.

PSO algorithm: Initialization

- Fitness function $f : \mathbf{R}^m \rightarrow \mathbf{R}$
- Number of particles $n = 20, \ldots, 200$
- Particle positions $x_i \in \mathbf{R}^m, \quad i = 1, \ldots, n$
- Particle velocities $v_i \in \mathbf{R}^m, \quad i = 1, \ldots, n$
- current best of each particle ("simple nostalgia") $\hat{x}_i$
- global best ("group norm") $\hat{g}$
- initialize constants $\omega, \ \alpha_{1/2}$

The canonical PSO algorithm

For each particle $1 \leq i \leq n$

- create random vectors

  r_1, r_2 with components drawn from $U[0,1]$

- update velocities

  $$v_i \leftarrow \omega v_i + \alpha_1 r_1 \circ \left(\hat{x}_i - x_i\right) + \alpha_2 r_2 \circ \left(\hat{g} - x_i\right)$$

- update positions

  $$x_i \leftarrow x_i + v_i$$

- update local bests

  $$\hat{x}_i \leftarrow x_i \quad \text{if} \quad f\left(x_i\right) < f\left(\hat{x}_i\right)$$

- update global best

  $$\hat{g} \leftarrow x_i \quad \text{if} \quad f\left(x_i\right) < f\left(\hat{g}\right)$$

$\circ$ componentwise multiplication

minimization problem!

Initialization

```
# Initialize the particle positions and their velocities
X = lower_limit + (upper_limit - lower_limit) *
                                        rand(n_particles, m_dimensions)
assert X.shape == (n_particles, m_dimensions)
V = zeros(X.shape)

# Initialize the global and local fitness to the worst possible
fitness_gbest = inf
fitness_lbest = fitness_gbest * ones(n_particles)
w=0.1                               # omega range 0.01 … 0.7
a1=a2=2                             # alpha range 0 … 4, both equal
n=25                                # range 20 … 200
max velocity                        # no larger than: range of x per step
                                           or 10-20% of this range
```

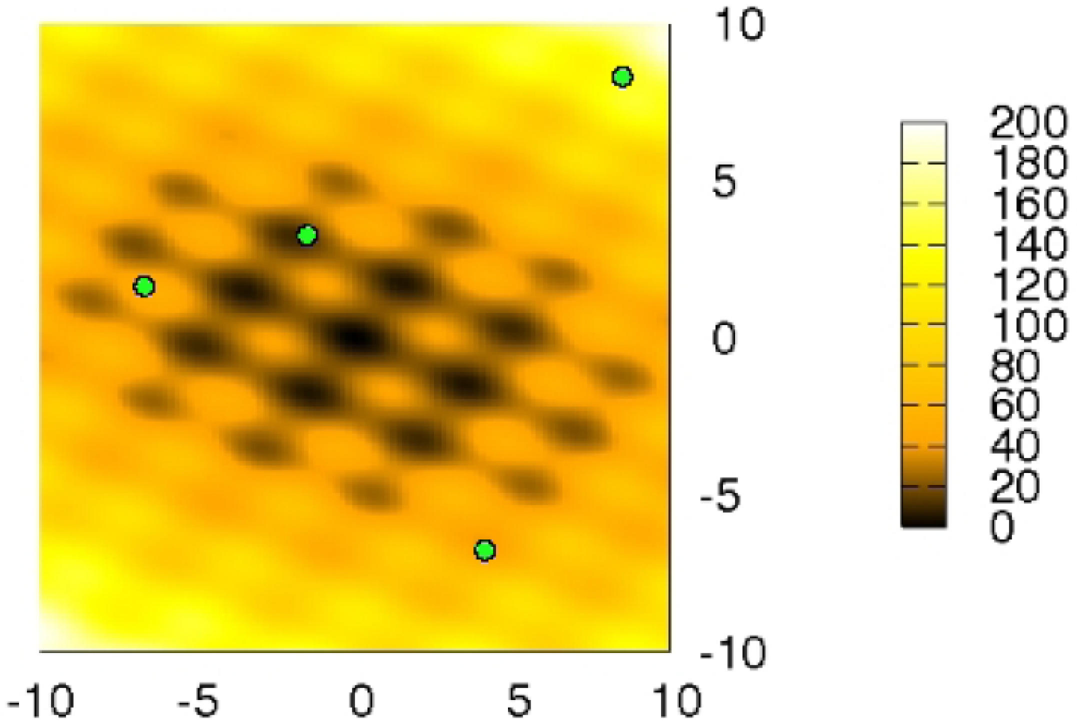
Main loop (next page)

```
for k in range(0, T_iterations):          # loop until convergence
  fitness_X = evaluate_fitness(X)         # evaluate fitness of each particle
  for I in range(0, n_particles):         # update local bests
    if fitness_X[I] < fitness_lbest[I]:
      fitness_lbest[I] = fitness_X[I]
      for J in range(0, m_dimensions):
        X_lbest[I][J] = X[I][J]
  min_fitness_index = argmin(fitness_X)         # update global best
  min_fitness = fitness_X[min_fitness_index]
  if min_fitness < fitness_gbest:
    fitness_gbest = min_fitness
    X_gbest = X[min_fitness_index,:]
  for I in range(0, n_particles):         # update velocities and positions
    for J in range(0, m_dimensions):
      R1 = uniform_random_number()
      R2 = uniform_random_number()
      V[I][J] = (w*V[I][J] +
             a1*R1*(X_lbest[I][J] - X[I][J]) + a2*R2*(X_gbest[J] - X[I][J]))
      X[I][J] = X[I][J] + V[I][J]
  end J,I,k; end;
```

Illustrative example

Marco A. Montes de Oca
PSO Introduction

1. Create a 'population' of agents (called *particles*) uniformly distributed over $\mathcal{X}$.



Repulsive PSO algorithm

For each particle $1 \leq i \leq n$

- create random vectors

  r_1, r_2, r_3 with components drawn from $U[0,1]$

- update velocities

$$v_i \leftarrow \omega v_i + \alpha_1 r_1 \circ (\hat{x}_i - x_i) + \alpha_2 r_2 \circ (\hat{y} - x_i) + \alpha_3 \omega r_3 \circ z$$

- update positions etc.

$\circ$ componentwise multiplication

$\hat{y}$ best of a random neighbor, $\alpha_2 < 0$

z random velocity

- Properties: sometimes slower, more robust and efficient

Constriction factor

- Introduced by Clerc (1999)
- Simplest form:

$$v_i \leftarrow \ K[\omega v_i + \alpha_1 r_1 \circ \left(\hat{x}_i - x_i\right) + \alpha_2 r_2 \circ \left(\hat{g} - x_i\right)]$$

$$K = \frac{2}{\left|2 - \varphi - \sqrt{\varphi^2 - 4\varphi}\right|}, \text{ where } \varphi = \alpha_1 + \alpha_2 > 4$$

e.g. $\varphi = 4.1 \ \Rightarrow \ K = 0.729$, i.e. prefactors $\alpha \approx 1.5$

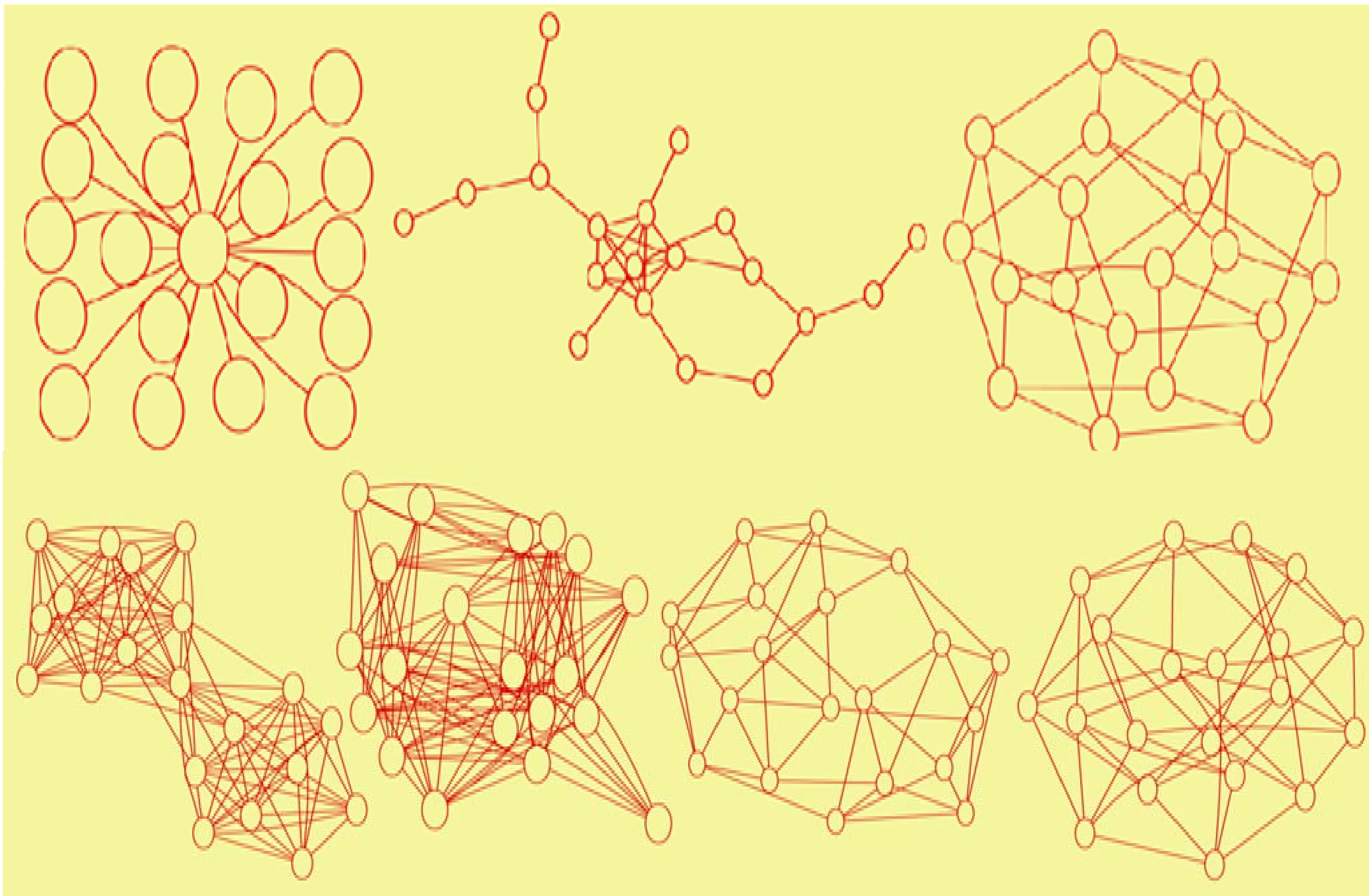
- May replace interia ω
- Meant to improve convergence by an enforced decay (more about this later)

Topology

- Topology determines with whom to compare and thus how solutions spread through the population
- Traditional ones: gbest, lbest
- Global version is faster but might converge to local optimum for some problems.
- Local version is a somewhat slower but not easy to be trapped into local optimum.
- Combination: Use global version to get rough estimate. Then use local version to refine the search.

Innovative topologies

- Specified by:
  Mean degree, clustering, heterogeneity etc.



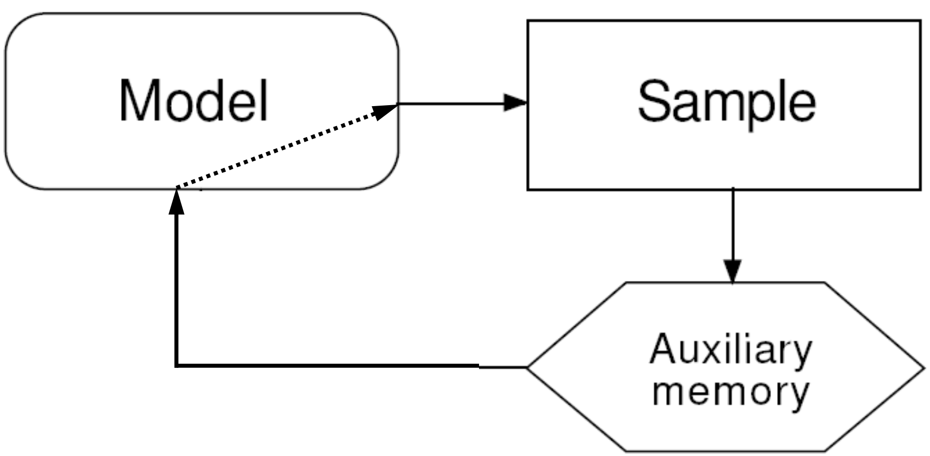
Comparison of GA and PSO

- Generally similar:
  1. Random generation of an initial population
  2. Caclulate of a fitness value for each individual.
  3. Reproduction of the population based on fitness values.
  4. If requirements are met, then stop. Otherwise go back to 2.
- Modification of individuals
  - In GA: by genetic operators
  - In PSO: Particles update themselves with the internal velocity. They also have memory.
- Sharing of information
  - Mutual In GA. Whole population moves as a group towards optimal area.
  - One-way in PSO: Source of information is only gBest (or lBest). All particles tend to converge to the best solution quickly.
- Representation
  - GA: discrete
  - PS: continuous

www.swarmintelligence.org/tutorials.php

PSO as MBS

- As in GA the “model” is actually a population (which can be represented by a probabilistic model)
- Generate new samples from the individual particles of the previous iteration by random modifications
- Use memory of global, neighborhood or personal best for learning



Literature on swarms

- Eric Bonabeau, Marco Dorigo, Guy Theraulaz: Swarm Intelligence: From Natural to Artificial Systems (Santa Fe Institute Studies on the Sciences of Complexity) (Paperback) OUP USA (1999)
- J. Kennedy, and R. Eberhart, *Particle swarm optimization*, in Proc. of the IEEE Int. Conf. on Neural Networks, Piscataway, NJ, pp. 1942–1948, 1995.
- Y Shi, RC Eberhart (1999) Parameter selection in particle swarm optimization. Springer.
- Eberhart Y. Shi (2001) PSO: Developments, applications ressources. IEEE.
- www.engr.iupui.edu/~eberhart/web/PSObook.html
- Tutorials: www.particleswarm.info/
- Bibliography: icdweb.cc.purdue.edu/~hux/PSO.shtml