Bio-Inspired Algorithms: An Introduction

Heitor Baldo

IMECC - Winter School
University of Campinas

July 11th - 15th, 2016

Course Summary

- Lecture 1 – Introduction, Evolutionary Computing & Genetic Algorithms
- Lecture 2 – Collective Intelligence
- Lecture 3 – Cellular Automata
- Lecture 4 – Artificial Immune Systems
- Lecture 5 – Neural Computation

Lecture 2 – Collective Intelligence

Collective Intelligence

Collective intelligence is problem-solving ability that resides in a *group* and is not present in, nor dictated by, any of its members [8, 10].

Three families of mechanism:

- **Aggregation**: Independent estimates are pooled statistically (crowd averaging, ensembles, voting).
- **Interaction**: Agents influence each other locally; the group state is a dynamical attractor (flocks, markets, opinion dynamics).
- **Stigmergy**: Agents modify a shared environment which in turn modifies their behaviour (ant trails, termite mounds).

For the mechanism to function, we must have: **diversity** of individual estimates or behaviours; **independence** enough to avoid herding; **decentralisation** (local knowledge is used locally); and **aggregation** (a mechanism that combines the parts).

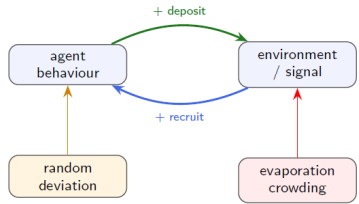
Removing independence, the same feedback that generated intelligence produces information cascades, technological lock-in, and collective error [12].

Self-Organisation: The Four Components

The four necessary components for self-organization are [8, 10]:

1. **Positive feedback** (amplification): Recruitment, reinforcement: a small initial advantage grows (*creates* structure).
2. **Negative feedback**: Saturation, exhaustion, competition, evaporation (*stabilises* the structure and prevents runaway).
3. **Fluctuations**: Randomness in behaviour (noise is the source of *innovation*).
4. **Multiple interactions**: A minimum density of agents sharing information directly or through the environment.

Positive feedback in blue/green, negative in red, fluctuation in orange.



Stigmergy

The term **stigmergy** was introduced by **Grassé** [1] to describe termite nest construction, referring to the principle that *completed work influences and stimulates subsequent activity*.

Two types of stigmergy:

- **Sematectonic**: The stimulus is the physical state of the task itself (a half-built pillar tells the termite to add material).
- **Sign-based**: The stimulus is a dedicated signal with no other function (a pheromone trail).

Features: No direct agent-to-agent messaging, hence no addressing, no broadcast storms; the environment acts as a shared, persistent, decaying memory; agents may be added or removed at any time.

The environment as memory: A pheromone field $\tau(t)$ obeying

$$\frac{\partial \tau}{\partial t} = -\rho\,\tau + \sum_k \Delta_k(t)$$

is an exponentially discounted record of past decisions. Evaporation rate ρ sets the memory horizon $1/\rho$ (short memory tracks a changing world, long memory exploits a stable one).

Emergence and Levels of Description

A property is **emergent** when it is a property of the collective that is not a property of any member individually, and is not explicitly represented anywhere in the system [8].

Levels of description (ant colony example):

Level	Ant colony	Algorithm
micro	individual ant rules	transition probability p_{ij}
meso	trail formation	pheromone field τ
macro	shortest path chosen	converged tour

The micro rules are the *program*, the macro behaviour is the *result*. Designing the first to obtain the second is the inverse problem at the core of swarm engineering, and it is hard, because emergence is by definition not compositional.

"The swarm decides": *which* local rule produces the decision? "Self-organisation solves it": what is the negative feedback that stops it? "Robust by design": robust to *which* perturbation, measured how?

Cellular automata (next class) are a good tool for studying emergence because the micro-rule is finite and completely specified.

Swarm Intelligence

Swarm intelligence can be defined as the emergent collective intelligence of groups of simple agents interacting locally with one another and with their environment, without central control [8].

Design principles:

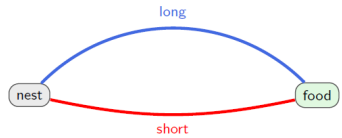
- Decentralised control, no leader, no global map;
- Local sensing and local action only;
- Stigmergic or short-range communication;
- Positive and negative feedback;
- Fluctuations enabling exploration;
- Robustness to the loss of individuals.

Engineering consequences: *Scalability*: cost per agent is independent of swarm size. *Fault tolerance*: no single point of failure. *Flexibility*: the same rules cope with a changed environment. *Cheapness*: agents can be simple and expendable.

Inspired by ant colonies, bird flocks, fish schools and bee swarms [7, 13, 5].

Case Study I – Ant Foraging and the Double Bridge

Deneubourg's experiment [3]: a nest is connected to a food source by two branches of different length. Ants initially choose at random, and after a few minutes almost all traffic uses the short branch.



Mechanism:

- Ants on the short branch return sooner, so pheromone accumulates there *faster* (a differential in *rate*, not in perception).
- Choice probability for branch A after n_A, n_B passages: $P_A = \dfrac{(k+n_A)^h}{(k+n_A)^h + (k+n_B)^h}$, with $h \approx 2$.
- Non-linearity $h > 1$ gives *autocatalysis* (symmetry breaking, then lock-in).

If both branches are equal, the colony still converges (on an arbitrary one). Positive feedback produces a decision, not necessarily the right one. Also, evaporation is what allows the colony to change its mind.

Case Study II – Honeybees

Foraging and the waggle dance [6]:

- A returning forager advertises direction, distance, and quality by dancing (dance *duration* is proportional to perceived profitability).
- Recruitment is therefore quality-proportional (exactly the sampling rule of fitness-proportionate selection (Lecture 1)).
- Unemployed bees follow dances (scouts search at random). Employed / onlooker / scout is the division of labour copied by the Artificial Bee Colony (ABC) algorithm.

Nest-site selection:

- Scouts inspect candidate cavities and dance for them. Support accumulates for the best sites and decays for the others.
- A quorum at one site (not a majority vote) triggers the swarm to move.
- Cross-inhibition between advocates of rival sites gives a decision rule mathematically close to a drift-diffusion model.

Case Study III – Flocks, Schools and Boids

Reynolds [2] reproduced flocking with three local rules applied to each agent within a limited perception radius, namely:

1. **Separation**: Steer to avoid crowding local flockmates.
2. **Alignment**: Steer towards the average heading of neighbours.
3. **Cohesion**: Steer towards the average position of neighbours.

No leader, no global plan, and no representation of "the flock". Adding a weak attraction to a goal and repulsion from obstacles gives usable navigation.

Alignment alone, in the zero-noise limit, is the Vicsek model: a phase transition from disorder to collective motion as density increases or noise decreases.

Collective Behavior-Inspired Optimization Algorithms: ACO, PSO, ABC, Firefly Algorithms

Next, we'll discuss the most popular swarm intelligence algorithms:

- Ant Colony Optimization (ACO);
- Particle Swarm Optimization (PSO);
- Artificial Bee Colony (ABC);
- Firefly Algorithm.

Ant Colony Optimization (ACO) – Construction

Proposed by **M. Dorigo** [4] (see also [7, 11]). ACO simulates the foraging behaviour of ants. Ants deposit pheromones on paths; shorter paths accumulate more pheromone and attract more ants. This problem can be cast as a **construction graph**: solutions are paths, and ants build them one component at a time. Probabilistic transition rule (probabilistic path selection): ant k at node i moves to node j with probability:

$$p_{ij}^k = \frac{[\tau_{ij}]^\alpha \, [\eta_{ij}]^\beta}{\sum_{l \in \mathcal{N}_i^k} [\tau_{il}]^\alpha [\eta_{il}]^\beta}, \qquad j \in \mathcal{N}_i^k,$$

where

- τ_{ij}: pheromone level on edge (i, j) (*learned* desirability);
- $\eta_{ij} = 1/d_{ij}$: heuristic (*a priori* desirability);
- $\mathcal{N}_i^k$: feasible neighbourhood (memory of visited nodes);
- $\alpha = 0$: greedy stochastic heuristic; $\beta = 0$: pure reinforcement, rapid stagnation.

Typical values (Travelling Salesman Problem (TSP)): $\alpha = 1$, $\beta = 2$–5, $\rho = 0.5$, $m \approx n$ ants, τ_0 small and uniform.

The rule is a softmax over a product of two evidence sources (exactly the structure of a Bayesian update with a learned prior and a fixed likelihood).

ACO – Learning

Let L_k be the tour length of ant k and Q a constant. The **pheromone update** after all m ants have completed a solution:

$$\tau_{ij} \leftarrow (1-\rho)\,\tau_{ij} + \sum_{k=1}^{m} \Delta\tau_{ij}^{k}, \qquad \Delta\tau_{ij}^{k} = \begin{cases} Q/L_k, & \text{if } k \text{ used edge } (i,j), \\ 0, & \text{otherwise.} \end{cases}$$

Algorithm 1: Ant System [7]

Data: graph, m, α, β, ρ, Q
Result: best solution found

```
initialise τ_ij ← τ_0
while budget remains do
    for k ← 1 to m do
        build tour T_k using p_ij^k
        evaluate L_k
    update best-so-far
    τ ← (1 − ρ)τ                                   // evaporation
    foreach k do deposit Q/L_k on the arcs of T_k
```

Evaporation is *forgetting*: it bounds τ and lets the colony escape an early wrong commitment. Deposit is *reinforcement learning* with a reward inversely proportional to cost. Together they define a stochastic gradient on the distribution over solutions (the connection to estimation-of-distribution algorithms).

Example: Travelling Salesman Problem (TSP)

Here we present an example of ACO on a small TSP instance. Consider four cities $\{A, B, C, D\}$ with symmetric distances:

$$D = \begin{pmatrix} 0 & 2 & 9 & 10 \\ 2 & 0 & 6 & 4 \\ 9 & 6 & 0 & 3 \\ 10 & 4 & 3 & 0 \end{pmatrix}.$$

Initial pheromone $\tau_0 = 1$ on all edges, $\alpha = 1$, $\beta = 2$, $\rho = 0.5$.

With short edges receiving more pheromone over iterations, ACO converges toward the optimal tour $A \to B \to D \to C \to A$ with length $2 + 4 + 3 + 9 = 18$.

ACO – Variants

- **Elitist Ant System (Elitist AS)**: The best-so-far tour deposits extra pheromone every iteration.
- **Rank-based AS**: Only the best w ants deposit, weighted by rank.
- **Ant Colony System** [7]: Pseudo-random proportional rule: with probability q_0 take the arc maximising $\tau\eta^\beta$ (exploitation), otherwise sample; plus a *local* update $\tau_{ij} \leftarrow (1-\xi)\tau_{ij} + \xi\tau_0$ that diversifies within an iteration.
- **MAX–MIN AS** [9]: Pheromone confined to $[\tau_{\min}, \tau_{\max}]$, only one ant deposits, τ initialised at $\tau_{\max}$. The bounds are what prevent stagnation, and they make convergence provable.

Local search matters: Competitive ACO for the TSP is always hybridised (each constructed tour is improved by 2-opt or 3-opt before the update). The colony then learns over *local optima* rather than raw tours.

ACO – Scope, Theory, and Practice

Where it applies:

- Any problem with a natural *constructive* heuristic: TSP, quadratic assignment, vehicle routing, scheduling, sequential ordering, etc.
- Dynamic problems, e.g., network routing, where evaporation tracks changing loads.
- Subset problems, with pheromone on components rather than arcs.

What is known theoretically:

- For MAX–MIN AS and ACS with best-so-far update: the probability of finding the optimum tends to one, and the algorithm converges in solution value.
- Convergence *in value* does not bound the time; runtime results exist only for toy problems.
- ACO is closely related to stochastic gradient ascent on a parameterised distribution over solutions, and to cross-entropy methods.

Failure mode to watch: *Stagnation*: all ants build the same tour, τ has collapsed onto one path and exploration has ended. Diagnose by tracking the entropy of p_{ij}, not the best-so-far curve.

Particle Swarm Optimization (PSO)

Kennedy & Eberhart [5] introduced PSO inspired by the social behaviour of bird flocks and fish schools. Each particle represents a candidate solution that moves through the search space, influenced by its own best position and the swarm's best position. For particle i at position $\mathbf{x}_i$ with velocity $\mathbf{v}_i$, the velocity and position update rules are:

$$\mathbf{v}_i^{t+1} = \underbrace{w\,\mathbf{v}_i^t}_{\text{inertia}} + \underbrace{c_1 r_1 \odot (\mathbf{p}_i - \mathbf{x}_i^t)}_{\text{cognitive}} + \underbrace{c_2 r_2 \odot (\mathbf{g} - \mathbf{x}_i^t)}_{\text{social}}$$

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

$r_1, r_2 \sim \mathcal{U}(0,1)^d$ drawn independently *per component*; $\mathbf{p}_i$ personal best, $\mathbf{g}$ neighbourhood best.

What it is, mechanically: A swarm of damped, stochastically forced harmonic oscillators whose attractors are the best points found so far. No selection, no replacement (an individual is never deleted, only re-accelerated).

Note the contrast with Lecture 1: PSO keeps *all* individuals and moves them, whereas an EA *replaces* individuals. Memory lives in $\mathbf{p}_i$ instead of in the population composition.

Example – PSO Minimising the Sphere Function

Let's minimise the *sphere function* $f(\mathbf{x}) = \sum_{i=1}^{n} x_i^2$, which has a global minimum $f(\mathbf{0}) = 0$.

With $N = 20$ particles, $n = 2$, $w = 0.7$, $c_1 = c_2 = 1.5$, and positions initialised in $[-5, 5]^2$:

Iteration	**Best fitness**	**g position**
1	18.42	$(3.21, -3.06)$
10	3.71	$(1.12, -1.63)$
50	0.08	$(0.21, -0.20)$
100	$< 10^{-4}$	$\approx (0, 0)$

PSO quickly converges toward the global optimum through cooperative exploration.

PSO – Parameters and Stability

Inertia weight w

- $w \geq 1$: velocities grow, swarm explodes.
- w decreasing linearly $0.9 \to 0.4$ over the run: a schedule from exploration to exploitation.

Constriction: Instead of clamping, use

$$\chi = \frac{2}{\left|2 - \varphi - \sqrt{\varphi^2 - 4\varphi}\right|}, \quad \varphi = c_1 + c_2 > 4,$$

giving $\chi \approx 0.7298$ with $c_1 = c_2 = 2.05$. This guarantees bounded trajectories. Velocity clamping $|v_{ij}| \leq v_{\max}$. A blunt but effective stabiliser; $v_{\max} \approx 0.1$–0.2 of the range.

Deterministic stability condition: For the mean trajectory, convergence requires

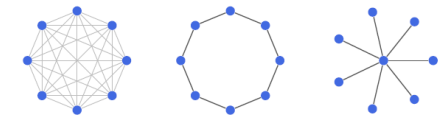
$$w < 1, \qquad 0 < c_1 + c_2 < 2(1 + w).$$

Outside this region the swarm diverges regardless of the objective.

Boundary handling (clamp, reflect, reinitialise) is not a detail: clamping to the box creates spurious attractors on the boundary.

PSO – Topology Is the Algorithm

The set of neighbours defining **g** determines how fast information spreads.



- **gbest**: One global attractor, information spreads in one step. Fast on unimodal problems, prone to premature convergence on multimodal ones.
- **lbest ring**: Information diffuses slowly, sub-swarms explore different basins. Slower but markedly more reliable.
- **von Neumann grid**: A good compromise, often the best default.

The code at the end of this lecture uses *gbest* and gets stuck at $f^* \approx 1.99$ on the same Rastrigin function that differential evolution solved to 10^{-6} in Lecture 1.

PSO – Variants

- **Binary PSO**: Velocity becomes a probability through a sigmoid, $P(x_{ij} = 1) = \sigma(v_{ij})$.
- **Discrete / permutation PSO**: Position updates redefined as sequences of swaps.
- **Fully informed particle swarm**: Every neighbour contributes, not just the best one.
- **Niching PSO**: Sub-swarms maintained around distinct optima for multimodal problems.
- **Multi-objective PSO**: An external archive of non-dominated solutions supplies the social attractor.

Artificial Bee Colony (ABC)

Karaboga's Artificial Bee Colony (ABC) algorithm [13] models the foraging behaviour of honey bees with three types of agents:

- **Employed bees:** Exploit known food sources; each food source is a candidate solution; employed bees perturb their own source by $v_{ij} = x_{ij} + \phi(x_{ij} - x_{kj})$, $\phi \sim \mathcal{U}(-1, 1)$ (a one-dimensional differential mutation); share information.
- **Onlooker bees:** Choose sources based on information from employed bees.
- **Scout bees:** Randomly explore new food sources when a source is exhausted.

Employed, onlooker and scout bees balance exploitation of known food sources with exploration of new ones.

Firefly Algorithm

Yang's Firefly Algorithm [14, 15] is inspired by the bioluminescent communication of fireflies. Fireflies are attracted to brighter ones (better solutions), with attraction decreasing with distance:

$$\beta(r) = \beta_0 \, e^{-\gamma r^2}.$$

The movement of firefly i towards the brighter firefly j is:

$$\mathbf{x}_i \leftarrow \mathbf{x}_i + \beta(r_{ij})(\mathbf{x}_j - \mathbf{x}_i) + \alpha \, \boldsymbol{\epsilon}_i,$$

where r_{ij} is the Euclidean distance between i and j, α is a randomisation coefficient, and $\boldsymbol{\epsilon}_i$ is random noise.

From Biology to Algorithm: the Common Pattern

Every swarm intelligence algorithm in this lecture instantiates the same template:

	Agent	**Shared memory**	**Positive feedback**	**Negative feedback**
ACO	ant building a tour	pheromone matrix τ	deposit $\propto 1/L$	evaporation ρ
PSO	particle with velocity	global best $\mathbf{g}$	attraction to $\mathbf{p}_i, \mathbf{g}$	inertia $w < 1$, clamping
ABC	employed/onlooker bee	food-source list	recruitment $\propto$ nectar	abandonment after *limit*
Firefly	firefly	perceived brightness	attraction $\beta e^{-\gamma r^2}$	randomisation α

The recurring question: Given a target macroscopic behaviour, what local rule produces it? There is no general constructive answer. In practice, one designs by analogy and validates empirically (or *evolves* the local rule with the methods of Lecture 1).

Python – Example: Particle Swarm Optimization

```python
import numpy as np
rng = np.random.default_rng(7)

def rastrigin(X):
    return 10 * X.shape[1] + np.sum(X**2 - 10 * np.cos(2 * np.pi * X), axis=1)

def pso(f, d=10, n=40, iters=1500, w0=0.9, w1=0.4, c1=1.7, c2=1.7,
        lo=-5.12, hi=5.12):
    X = rng.uniform(lo, hi, (n, d))                 # positions
    V = rng.uniform(-1, 1, (n, d)) * (hi - lo) * .1  # velocities
    P, fP = X.copy(), f(X)                          # personal bests
    g = P[np.argmin(fP)].copy(); fg = fP.min()      # global best
    vmax = 0.2 * (hi - lo)
    for t in range(iters):
        w = w0 + (w1 - w0) * t / iters              # linearly decreasing inertia
        r1, r2 = rng.random((n, d)), rng.random((n, d))
        V = w * V + c1 * r1 * (P - X) + c2 * r2 * (g - X)
        V = np.clip(V, -vmax, vmax)
        X = np.clip(X + V, lo, hi)
        fX = f(X)
        imp = fX < fP                               # update personal bests
        P[imp], fP[imp] = X[imp], fX[imp]
        if fP.min() < fg:                           # update global best
            fg, g = fP.min(), P[np.argmin(fP)].copy()
    return g, fg

x, fx = pso(rastrigin)
print("f* = %.6f" % fx)
```

Output: `f* = 1.989918`. Two coordinates are stuck one full period away from the optimum: classic *gbest* premature convergence.

Python – Example: Ant System for the TSP

```python
import numpy as np
rng = np.random.default_rng(3)

N = 30                                          # cities
C = rng.random((N, 2))
D = np.linalg.norm(C[:, None, :] - C[None, :, :], axis=2) + np.eye(N) * 1e-9
eta = 1.0 / D

def tour_length(t):
    return D[t, np.roll(t, -1)].sum()

def ant_system(m=30, iters=300, alpha=1.0, beta=5.0, rho=0.5, Q=1.0):
    tau = np.ones((N, N)) / N                   # pheromone matrix
    best, best_len = None, np.inf
    for it in range(iters):
        tours = []
        for k in range(m):
            unvisited = list(range(N))
            i = int(rng.integers(N)); unvisited.remove(i); t = [i]
            while unvisited:
                u = np.array(unvisited)
                p = (tau[i, u] ** alpha) * (eta[i, u] ** beta)
                p = p / p.sum()                 # probabilistic transition rule
                i = int(rng.choice(u, p=p)); unvisited.remove(i); t.append(i)
            tours.append(np.array(t))
```

Python – Example: Ant System for the TSP II

```
        L = np.array([tour_length(t) for t in tours])
        if L.min() < best_len:
            best_len, best = L.min(), tours[int(np.argmin(L))]
        tau *= (1 - rho)
        for t, l in zip(tours, L):
            tau[t, np.roll(t, -1)] += Q / l
            tau[np.roll(t, -1), t] += Q / l
    return best, best_len

t, l = ant_system()
print("best tour length = %.4f" % l)
```

Output on a random 30-city instance in the unit square: `best tour length = 4.6361`.

Baselines on the same instance: best nearest-neighbour tour 5.0007; mean random tour 14.87. The colony beats the greedy heuristic by about 7% without any local search.

Homework (Lecture 2)

1. On PSO:

- Replace $\mathbf{g}$ by a ring topology local best (each particle sees $i \pm 1$). Does the $f^* = 1.99$ trap disappear?
- Fix $w = 0.7298$, $c_1 = c_2 = 1.49618$ (constriction) instead of the linear schedule.
- Remove velocity clamping and watch the swarm explode.

2. On ACO:

- Set $\beta = 0$ (no heuristic) and then $\alpha = 0$ (no pheromone). Which degrades more?
- Add 2-opt to every constructed tour before the update (expect a large jump in quality).
- Implement MAX–MIN bounds $[\tau_{\min}, \tau_{\max}]$ and let only the iteration-best ant deposit.
- Move one city halfway through the run: how large must ρ be for the colony to adapt?

References I

[1] P.-P. Grassé. *La reconstruction du nid et les coordinations interindividuelles chez Bellicositermes natalensis et Cubitermes sp.: la théorie de la stigmergie.* Insectes Sociaux, 6:41–80, 1959.

[2] C. W. Reynolds. *Flocks, herds and schools: a distributed behavioral model.* Computer Graphics (SIGGRAPH '87), 21(4):25–34, 1987.

[3] J.-L. Deneubourg, S. Aron, S. Goss, and J. M. Pasteels. *The self-organizing exploratory pattern of the Argentine ant.* Journal of Insect Behavior, 3(2):159–168, 1990.

[4] M. Dorigo. *Optimization, Learning and Natural Algorithms.* Ph.D. thesis, Politecnico di Milano, 1992.

[5] J. Kennedy and R. Eberhart. *Particle swarm optimization.* Proc. IEEE Int. Conf. on Neural Networks, 1942–1948, 1995.

[6] T. D. Seeley. *The Wisdom of the Hive: The Social Physiology of Honey Bee Colonies.* Harvard University Press, 1995.

[7] M. Dorigo, V. Maniezzo, and A. Colorni. *The Ant System: optimization by a colony of cooperating agents.* IEEE Trans. Systems, Man, and Cybernetics–B, 26(1):29–41, 1996.

[8] E. Bonabeau, M. Dorigo, and G. Theraulaz. *Swarm Intelligence: From Natural to Artificial Systems.* Oxford University Press, 1999.

References II

[9] T. Stützle and H. H. Hoos. *MAX–MIN Ant System.* Future Generation Computer Systems, 16(8):889–914, 2000.

[10] S. Camazine, J.-L. Deneubourg, N. R. Franks, J. Sneyd, G. Theraulaz, and E. Bonabeau. *Self-Organization in Biological Systems.* Princeton University Press, 2001.

[11] M. Dorigo and T. Stützle. *Ant Colony Optimization.* MIT Press, 2004.

[12] J. Surowiecki. *The Wisdom of Crowds.* Doubleday, 2004.

[13] D. Karaboga. *An idea based on honey bee swarm for numerical optimization.* Technical Report TR06, Erciyes University, 2005.

[14] X.-S. Yang. *Nature-Inspired Metaheuristic Algorithms.* Luniver Press, 2008.

[15] X.-S. Yang. *Firefly algorithms for multimodal optimization.* Stochastic Algorithms: Foundations and Applications (SAGA 2009), Lecture Notes in Computer Science 5792, 169–178, Springer, 2009.