Genetic Algorithms and Genetic Programming

Lecture 7: (16/10/09)

Evolutionary algorithms (in Robotics)

Michael Herrmann

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

Overview

1. Introduction: History
2. The genetic code
3. The canonical genetic algorithm
4. Examples & Variants of GA
5. The schema theorem
6. Hybrid algorithms
7. Evolutionary robotics
8. GP

Hybrid GA: **Evolving Neural Networks**

- Reminder of neural networks
- Evolving weights
- Evolving network topology 
- Grammars, robotics
- Evolving intelligent behaviours
- Example: evolving communication

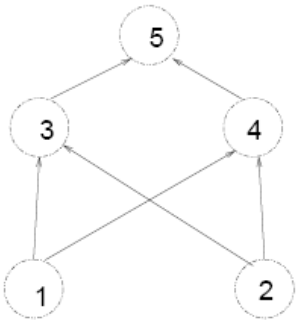
Evolving Topology 1

– choosing a network topology is hard

– can it be done automatically?

Miller, Todd and Hegde (1989):

from unit:		1	2	3	4	5
to unit:	1	0	0	0	0	0
	2	0	0	0	0	0
	3	1	?	?	?	?
	4	1	?	?	?	?
	5	0	?	?	?	?



0 if not connected
1 if connected = learnable
? = you complete the table

Chromosome: 00000 00000 ... (complete the rest...) **Mutation**: bit flipping

Crossover: exchange whole rows Limit to **feedforward networks**: any links to input units or feedback connections are ignored.

Evolving Topology 2

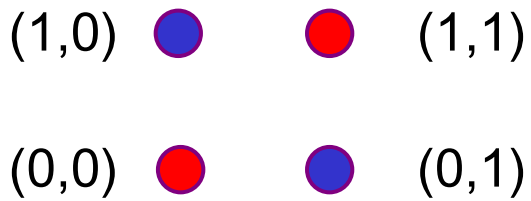
Tasks tried by Miller et al.:

(a) XOR (exclusive - OR)

(b) four quadrant:

(x,y) $\rightarrow 0.0$ if $x, y \simeq 0.0$ or $x, y \simeq 1.0$

(x,y) $\rightarrow 1.0$ otherwise



(c) pattern copying, with units in the hidden layer $<$ number of input units

Learning: back-propagation

Results: GA can easily find network topologies for these problems.

But are the problems too easy?

See Stanley and Miikkulainen (2002) for a more sophisticated approach (NEAT)

NEAT Neuro-Evolution through Augmenting Topologies

• Evolve by changing the connection weights, turning links on and off, and also by adding nodes and links (in the mutation stage)

Start off simple, become more complex – as complex as needed – complexification

If a node is added it is added in the middle of an existing link

• Crossover: need to match up parts of the network coding for similar traits.

Competing conventions: permuting a network doesn't change the outputs or the function computed by the network

So: give each gene an $innovation\ number$ – the next unused integer when it is made. So can match up parts of networks inheriting this gene in future generations.

NEAT Neuro-Evolution through Augmenting Topologies

Inherit matching genes from each parent with equal probability. Inherit non-matching genes from fittest parent.

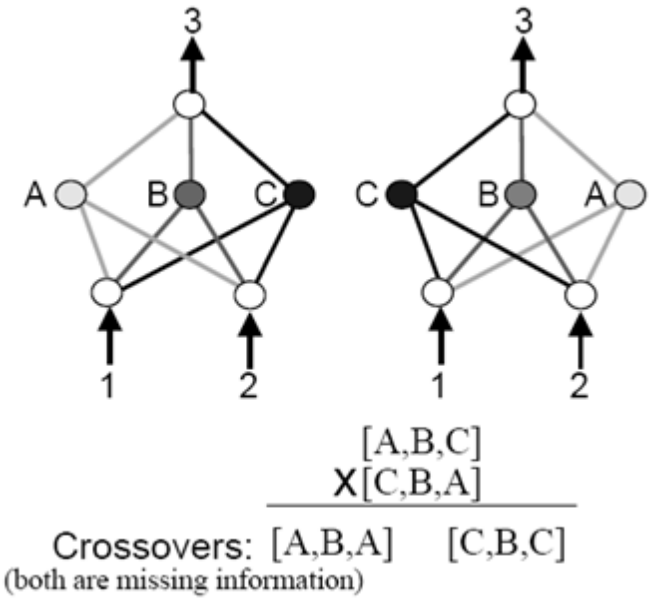
- Can also split into species based on difference between chromosomes (based on number of matching genes and other metrics). Preserves new topology for a while so that it has a chance to optimise its structure only in competition with similar members.
- Works well on pole-balancing. Also applied to game of GO.

See Kenneth O. Stanley and Risto Miikkulainen. Evolving neural networks through augmenting topologies. Evolutionary Computation 10, 99–127 (2002)

Risto Miikkulainen

Kenneth O. Stanley

Evolutionary Computation 10(2): 99-127



On using “Innovation numbers”

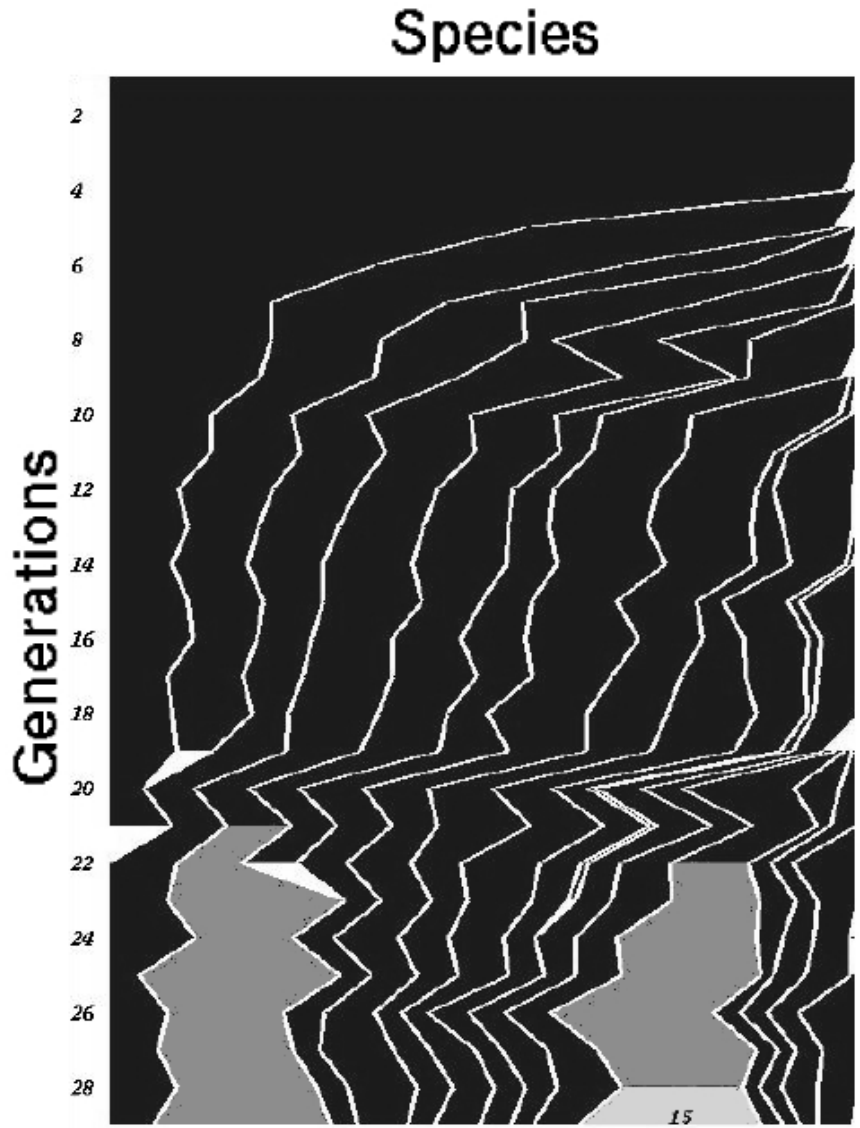
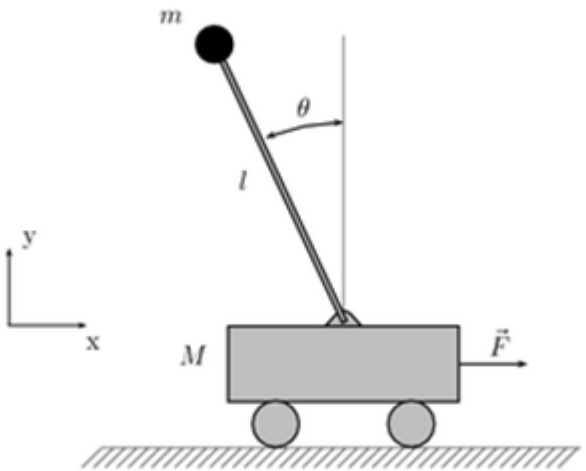
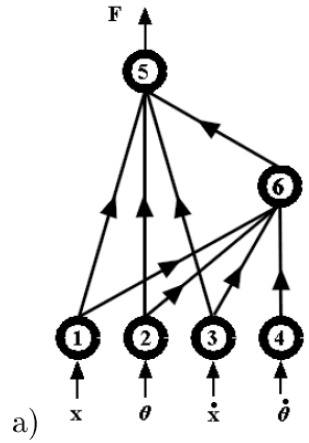


Figure 7: Visualizing speciation during a run of the double pole balancing with velocity information task. Two species begin to close in on a solution soon after the 20th generation. Around the same time, some of the oldest species become extinct.

Evolving Neuromodules for Control





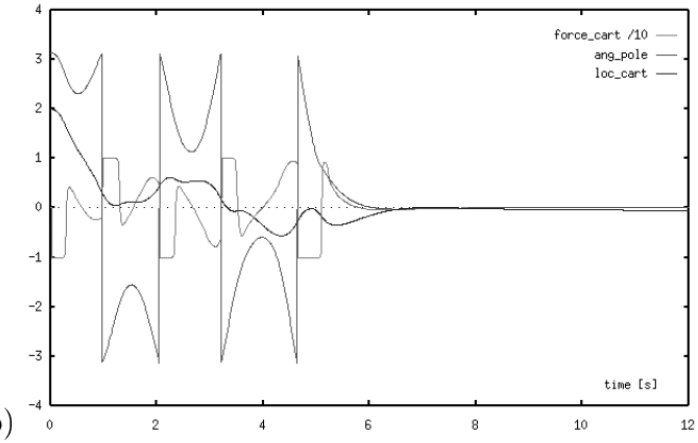
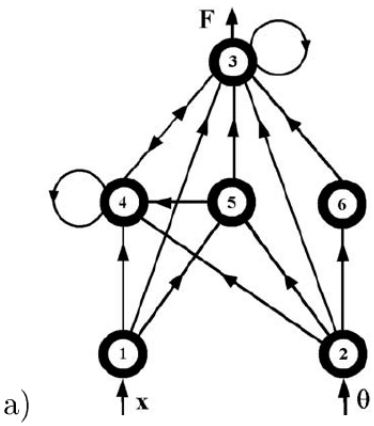


Figure 1: a.) A minimal 4t-class solution w^1 and b.) its effective control: $x(t)$, $\theta(t)$, and $F(t)$ starting from $x_0 = 2.0$ and $\theta_0 = \pi$.



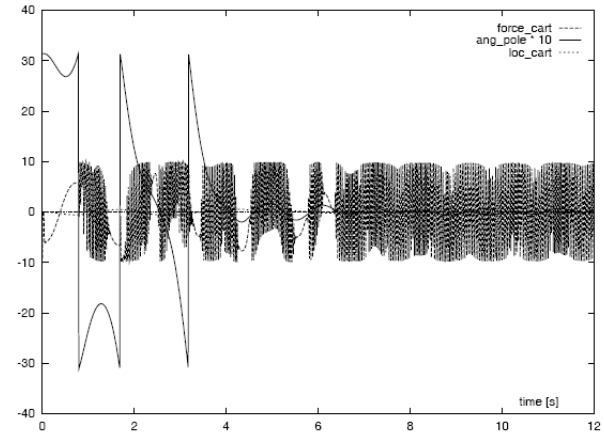


Figure 5: a.) The 2t-controller w^3 solving the swinging-up problem, and b.) cart position and pole angle under its action, starting from $x_0 = 0$, $\theta_0 = \pi$.

F. Pasemann et al. (1999) Evolving structure and function of neurocontrollers. MIS preprint

Evolutionary Computation

- Formalization of GA
- Four types of EC: GA, GP, EP, ES
- Simple evolutionary strategies (ES)
- Self-adapting ES
- Nested ES
- Applications of EC

Beyond the Schema theorem: Formalization of GAs

- Canonical GA, binary encoding, fitness proportional selection, only one offspring from each crossover
- Search space indexed by i=0,…, 2^{ℓ}-1, ℓ: string length
- $p_i(t)$ proportion of the population consisting of string i
- $s_i(t)$ probability that i is selected as a parent
- $r_{ij}(k)$ probability that k is produced by crossover of i and j
- f_i (fitness of i) *is used as a matrix* $F_{ii}=f_i$ *and* $F_{ij}=0$ *for* $i \neq j$

$$\vec{s}(t) = \frac{F\vec{p}}{\sum_{j=0}^{2^{\ell}-1} F_{jj} p_j(t)} \qquad E\left(p_k(t+1)\right) = \sum_{i,j} s_i \, s_j \, r_{i,j}(k)$$

$$\vec{s}(t+1) = G\vec{s}(t) \iff \vec{p}(t+1) = G_p(\vec{p}(t))$$

for large populations

- GA Operator composed of operators for selection, mutation etc.
- Further details in Mitchell p. 139, Whitley tutorial p. 79

Vose, M. and Liepins, G. (1991). Punctuated equilibria in genetic search. *Complex Systems,* 5, 31-44.

Results of the Formalization

- Operator formalism:
  Operators for selection, mutation, crossover
- Dynamical systems formulation:
  Iteration, fixed points, stability
  Assumes infinite populations
- Finite-population theory: Markov chains
- Statistical mechanics approach: Assume a particular (“Boltzmann”) selection scheme:
  Prediction of “macroscopic” properties of GA
  (Shapiro, Prügel-Bennet, 1994)

Evolutionary algorithms

	genotype (encoding)	mutation/ crossover	phenotype (applied to)
Genetic algorithm	strings of binary or integer numbers	e.g. 1-point for both with p_m, p_c	optimization or search of optimal solutions
Genetic programming	strings of binary or integer numbers	e.g. 1-point for both with p_m, p_c	computer programs for a computational problem
Evolutionary programming	real numbers	mutation with self-adaptive rates	parameters of a computer program with fixed structure
Evolution strategy	real numbers	mutation with self-adaptive rates	optimization or search of optimal solutions

Evolution Strategies

- I. Rechenberg, H.-P. Schwefel (1970s)
- Population-based real-valued optimisation
- Self-adaptation of (mutation) parameters standard: individuals contain problem-dependent code and parameters of the algorithms (e.g. mutation rates)
- Learning of correlations between mutation rates
- On the other hand: “greedy” selection and simple recombination (e.g. by averaging the parents)
- nested (hierarchical) algorithms [island algorithms]
- “comma” and “plus” variants [without or with elitism]

Evolution Strategies

$$y^* = \arg opt_{y \in Y} f(y), \quad y \in R^L$$

Canonical ESs: (μ/ρ,λ)-ES or (μ/ρ+λ)+ES

μ: populations size after selection (potential parents)
ρ: number of parents (mixing number) ρ < μ
λ: number of offspring (population size)

„**comma**“ selection: new population is selected deterministically from the offspring μ < λ

„**plus**“ selection: new population is selected deterministically from the parent generation and the offspring μ < λ (similar to elitism)

http://www.scholarpedia.org/article/Evolution_strategies

Self-Adaptation Evolution Strategy

1. Initialize parent population $P_\mu = \{a_1, , a_\mu\}$, each consisting of (y,s)
2. Generate λ offspring *a'* forming the offspring population $P_\lambda' = \{a_1', , a_\mu'\}$ where each offspring *a'* is generated by:
   a. Select (randomly) ρ parents from P_μ (if ρ=μ take all parental individuals instead).
   b. Recombine the ρ selected parents a to form a recombinant individual *r*.
   c. Mutate the strategy parameter set s of the recombinant *r*.
   d. Mutate the objective parameter set *y* of the recombinant *r* using the mutated strategy parameter set to control the statistical properties of the object parameter mutation.
3. Select new parent population (using deterministic truncation selection) from either
   1. the offspring population P_λ' (this is referred to as *comma*-selection, usually denoted as (μ, λ)-selection), or
   2. the offspring P_λ' *and* parent P_μ population (this is referred to as *plus*-selection, usually denoted as (μ+λ) -selection)
4. Goto 2. until *termination criterion* fulfilled.

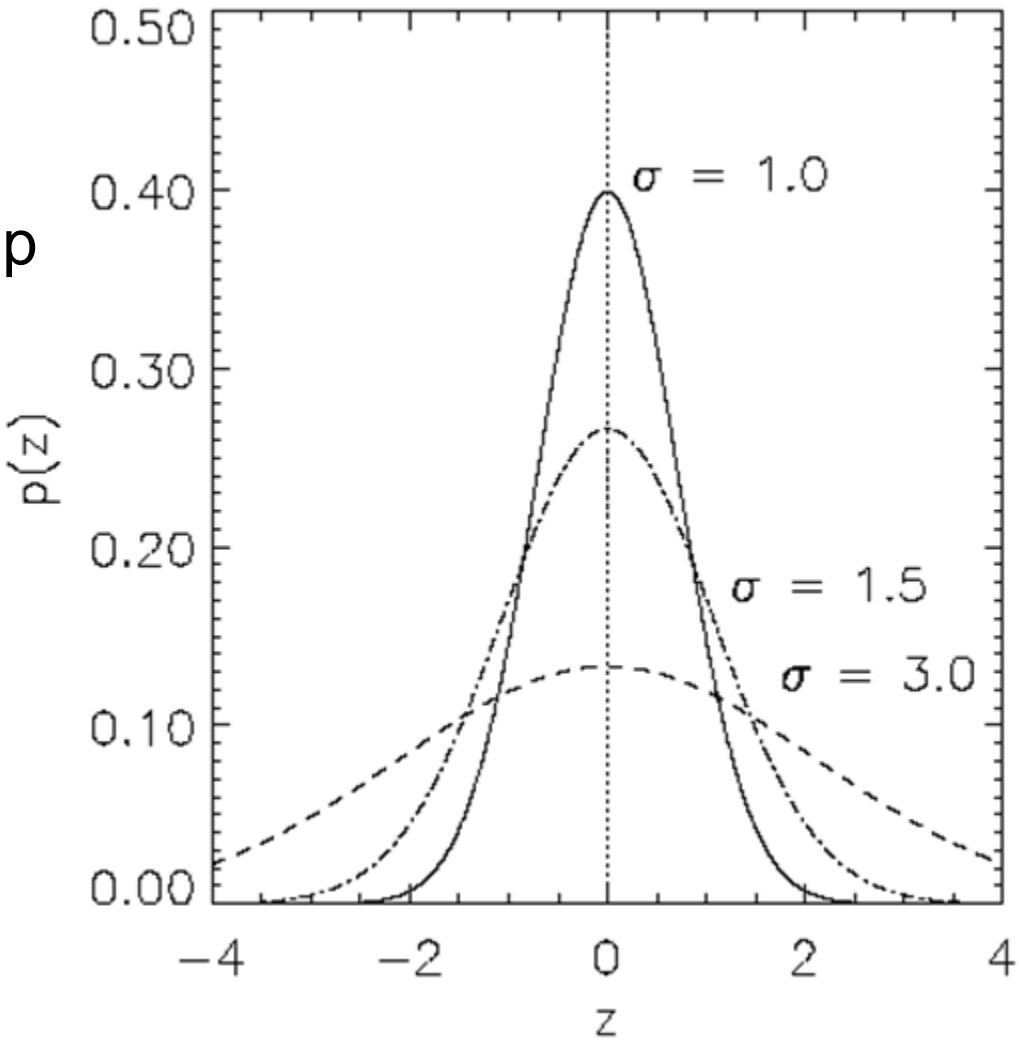
http://www.scholarpedia.org/article/Evolution_strategies

Genetic operators: Mutations

$y_i(t) = x_i(t) + z_i$

z values drawn from normal distribution $N(\xi,\sigma)$
mean ξ is set to 0
variation σ is called mutation step size
σ is varied on the fly by the "1/5 success rule":
This rule resets σ after every k iterations by
$\sigma = \sigma / c$ if ps > 1/5
$\sigma = \sigma \bullet c$ if ps < 1/5
$\sigma = \sigma$ if ps = 1/5
where p_s is the % of successful mutations, $0.8 \leq c \leq 1$

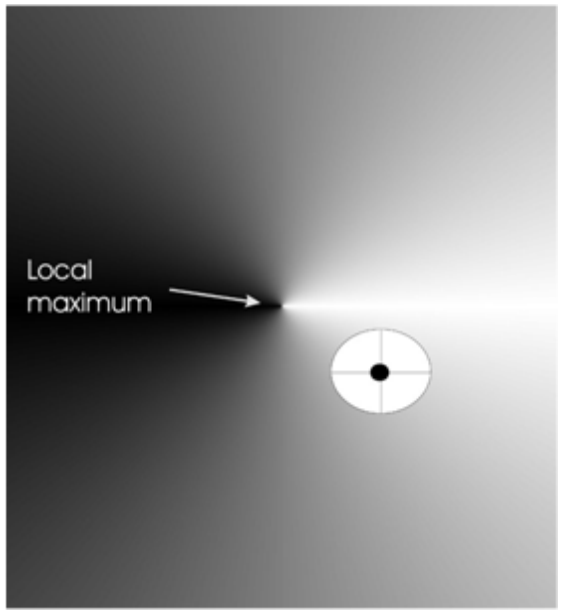
The one dimensional case



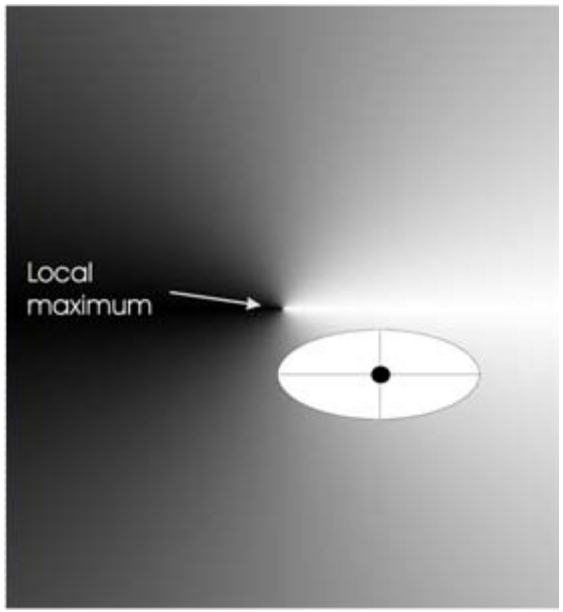
A.E. Eiben and J.E. Smith, Introduction to Evolutionary Computing. Evolution Strategies

Multidimensional mutations in ES

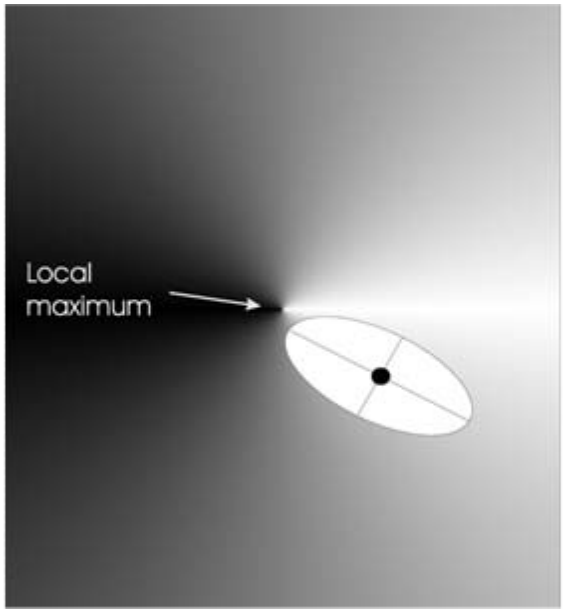
Uncorrelated mutation with one σ



Uncorrelated mutation with L σ_i's



Correlated mutations



Correlated mutations: $y = x + \mathcal{N}(0, C')$

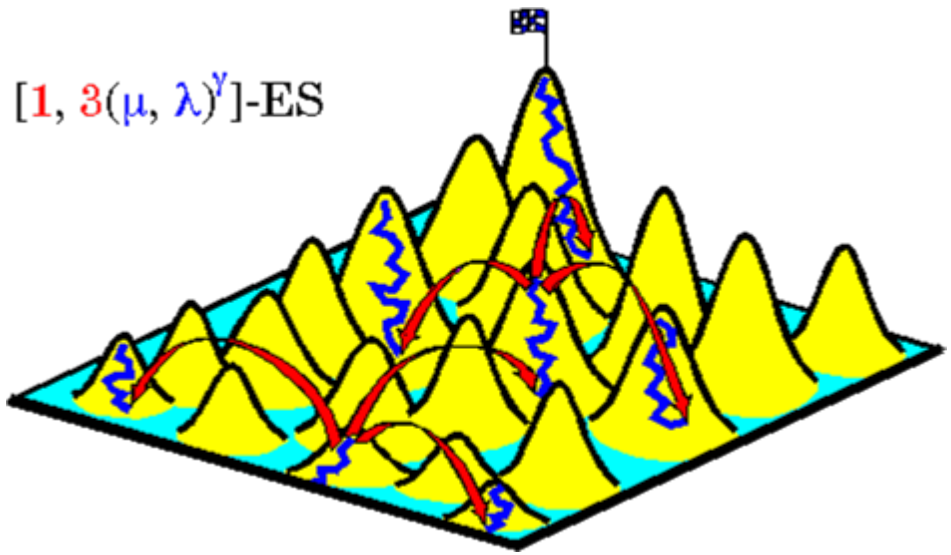
x stands for the vector $(x_1, \ldots, x_n)$

C' is the covariance matrix C after mutation of the σ values

A.E. Eiben and J.E. Smith, Introduction to Evolutionary Computing. Evolution Strategies

Nested Evolution Strategy

- Hills are not independently distributed (hills of hills)
- Find a local maximum as a start state
- Generate 3 offspring populations (founder populations) that then evolve in isolation
- Local hill-climbing (if convergent: increase diversity of offspring populations)
- Select only highest population
- Walking process from peak to peak within an “ordered hill scenery” named *Meta-Evolution*
- Takes the role of crossover in GA



http://www.bionik.tu-berlin.de/intseit2/xs2mulmo.html

Evolutionary Robotics

- What are robots? Possible answer: interfaces between computers and the real world
- Evolution of controllers
  http://www.youtube.com/watch?v=ehno85yI-sA
  http://www.youtube.com/watch?v=jMyVbFDzxes
- Hardware evolution is an option:
  - Evolvable hardware: Modular robots
    http://www.youtube.com/watch?v=AIjzMszshVM
  - The Golem project
    http://www.youtube.com/watch?v=sLtXXFw_q8c
  - Overview by
    K.C. Tan, L.F. Wang, T.H. Lee, P. Vadakkepat: **Evolvable Hardware in Evolutionary Robotics.** *Autonomous Robots* 2004

Evolutionary Robotics by Stefano Nolfi and Dario Floreano. ISBN 0-262-14070-5

Evolving neural networks for control

Grammars and Robotics

- Grammatical encoding of the linkage matrix (here for XOR)

$$S \rightarrow \begin{matrix} A & B \\ C & D \end{matrix} \rightarrow \begin{matrix} c & p & a & a \\ a & c & a & e \\ a & a & a & a \\ a & a & a & b \end{matrix} \rightarrow \begin{matrix} 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{matrix}$$

(S A B C D | A c p a c | B a a a e . . .)

Generate linkage matrix from the grammar. If at the end of rewriting there are still non-terminal nodes, that node is "dead" – not connected.

• Develop chromosome (genotype) into network (phenotype) and train for fixed no. of training episodes.

• Fitness = error at end of training

Problems with direct encoding

Fixed connections: as size of matrix grows, chromosome size grows

Can't encode repeated patterns, esp. with internal structure

Takes a long time to generate high-performing networks

Advantages of grammatical encoding

Can represent large connectivity matrices in compact form

Shorter encoding, faster search

Variable topologies including recurrent connections

Better on encoder/decoder problem than direct encoding

Evolving Neural Network Behaviours

- Previous examples rely on *training data*
- What if we haven't got any?
- Example: a neural network which controls a mobile agent which is trying to achieve some goals in a dynamic environment.
- No good example of behaviour is available; or we wish to try a range of possible behaviours to see which is best.
- A fitness function is available based on goal achievement.

Evolving Neural Network Behaviours

General approach:

- Decide on how to represent inputs to and outputs from the neural network.
- Decide on a neural network architecture: might need to try a range of possibilities.
- Decide on a simulation which tests the NN's behaviour.
- Decide on a fitness function which tests how well the NN did in the simulation.
- All the usual GA stuff: chromosome representation, crossover, mutation, population size, etc.

Example: Evolving Communication

This is an example from Artificial Life: the study of computer generated "life" forms. (Matthew Quinn, University of Sussex)

- Khepera robots controlled by evolved neural networks
- Group task: robots move together as far as possible – like dancing
- 8 sensor nodes, 4 motor nodes, hidden nodes
- Evolved thresholds, weights, decay parameters, size, connectivity of network
- **Co-evolution**: select two robots from population, rate them for fitness $as\ a\ pair$
- Initial result: leaders and followers emerge
- Only get a working pair 50% of the time, BUT....

Example: Evolving Communication

- After a while a new single species emerges
- This behaviour uses communication based on simple movement:
  - both agents (A and B) rotate anti-clockwise
  - one agent (B) becomes aligned first and moves towards the other agent
  - agent B moves backward and forward while staying close to A
  - when A becomes aligned, it becomes the leader: it reverses its direction and is followed by B
- Very similar to movement communication used in social insects (e.g. dancing in honey bees)

Evolving Robots Learn To Lie To Each Other

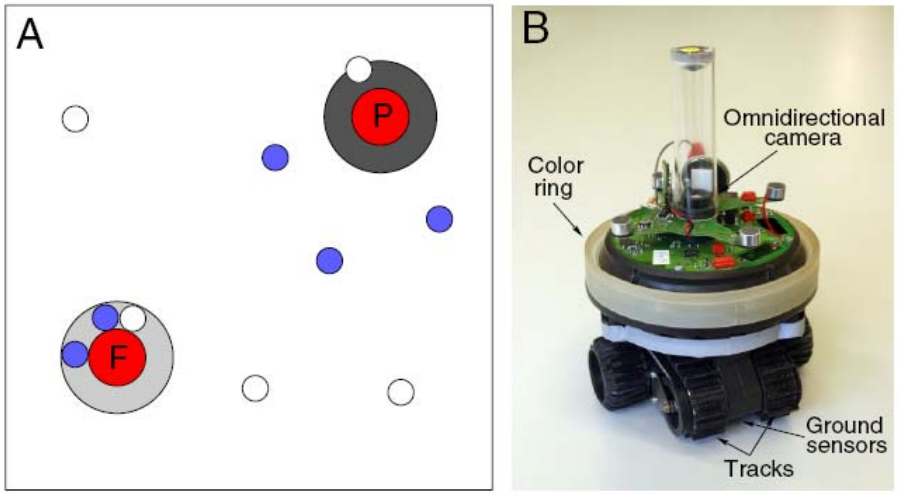


Fig. 1. Experimental setup. (*A*) A food and poison source, both emitting red light, are placed 1 m from one of two opposite corners of the square (3- × 3-m) arena. Robots (small circles) can distinguish the two by sensing the color of the circles of paper placed under each source by using their floor sensors when driving over the paper. (*B*) The robot used for the experiments is equipped with two tracks to drive, an omnidirectional (360°) vision camera, a ring of lights used to emit blue light, and floor sensors to distinguish food and poison sources (see ref. 14 for details).

- 1,000 robots divided into 10 groups
- Each robot had a sensor, a blue light, and a 264-bit binary genome encoding a controller
- initial population: turn the light on at food resource, supporitng also other robots
- positive fitness points for finding and sitting at the good resource, negative for being near the poison
- 200 fittest robots are selected, recombined and mutated
- Near optimal fitness after 9 generations
- A limited amount of food results in overcrowding

- After 500 generations: 60 % of the robots kept their light off near food
- Other robots adapted to this and developped an aversion to the light

Sara Mitria, Dario Floreano and Laurent Keller (2009) The evolution of information suppression in communicating robots with conflicting interests. PNAS

Outlook

- Genetic and Evolutionary programming