Disclaimer ☺

This document is a summary of Prof. Floreano's Bio-inspired Adaptive Machines course. The purpose is to help the student revise for the oral examination. This document should not be considered as a replacement for the course & course work. This document contains screen copies of the most important slides from the course website.
I am in no way to be held responsible for any mistakes or lack of information in this document.

Evolutionary Systems (Genetic algorithms GA)

Genetic algorithms encode solutions (phenotypes) as strings (genotypes), and provide operators on these strings in order to maximize the solution. GA are useful when you don't want the best solution, but a good solution.
GA is based on the reproduction of 2 (or more) parents using crossover and random mutations. In some cases, the crossover operation needs to be adapted to the problem (e.g.: salesman).

Exploration (go to unknown region to avoid local maxima) / Exploitation problem (improve individual towards local maxima):
Diversity can be increased by mutation & recombination => exploration
Selection of parents & survivors decrease the diversity => exploitation

There are multiple ways to select the parents and which individuals must be replaced (see slides). It is important that the recombination produce valid chromosomes. The child must also inherit something from both parents, and the recombination operator should be designed in conjunction with the representation (or otherwise the recombination is catastrophic).

Pros of GA:

- parallel processing
- no presumption on problem space (it's also a cons)
- widely applicable
- can be run interactively
- Low development & application cost!
- Provides many alternative solutions!

In order to have GA, you must have a way to rate a given solution (fitness function). The fitness function must be continuous.

Cellular Systems and Cellular Automata (CA)

The key idea is to build a complex system using simple cells that are replicated multiple times.

Modelling cellular systems is done using the following 4 concepts:

- Cellular space => Initial condition.
- Neighbourhood – local interaction. => Boundary conditions.
- Cell state
- Transition rules (represented by table or binary number (Wolfram's Rule Code))
  (totalistic => depends only on neighbours, outer totalistic => depends on self + neighbours)

The cell states are updated synchronously (at discrete time steps).

Applications: snow, traffic (rule 184), RNG (random number generator) (rule 30), game of life, computation (div by 2, rule 132), etc…

Extension of CA: Probabilistic CA (forest fire, epidemic), Particles (space is divided in blocks that alternate between even and odd space partition).

CA can also be used to analyse global properties (e.g.: patterns) based on local mechanisms.

Neural Systems

Two types of neural networks (NN): McCulloch-Pitts (based on firing rate) and Spiking neurons (based on firing time). The problem is we don't have any back-propagation equivalent for the Spiking neurons.

NN have 3 components: input layer, hidden layer and output layer. The function performed in each neuron must be continuous & monotonic (& bounded & horizontal asymptotes). The input layer can also contain a bias (additional input whose value is always -1)

The output of a neuron is a measure of how similar is its current input pattern to its pattern of connection weights.

Hebb rule (1949 Donald Hebb): connection weight should be strengthened whenever both the postsynaptic and presynaptic neurons are active. It is better to normalize the weights to avoid self-amplification. Such neurons tell how familiar a pattern is.

Supervised learning (aka delta rule, gradient descent) is a method to adjust the weights so that the error between the current output and a desired output is reduced.

Back propagation (Rumelhart et al. 1986): propagate the error back into the hidden layer.

If the neural network doesn't have a hidden layer (aka perceptrons), it can only solve linearly separable problems.
Problems with neural networks: local minima & over-fitting.

Applications: see course slides.

Behavioural Systems

Key principal: no planning. Sensors (stimulus) activate the reactions (response). The final robot is sometimes less predictable, but is very responsive and doesn't depend on an accurate world model (since the world itself is the model).

Brooks, 1986: Augmented finite state machine. No global clock!

Conflict resolution: multiple methods (suppression/inhibition, priority, action selection, vote based, fusion).

Usually the system is built incrementally (by adding new behaviours).

Evolutionary Robotics

Suggested by Braitenberg, 1984.

NN or GA is used to create behaviours. The input layer receives the values of the sensors, and the output layer controls the motors.

It is sometimes hard to evolve a complex robot. Experiments have proven that it is easier to evolve a simple robot and then to use the simpler robot's evolved behaviour on the more complex one. A few iterations will be needed to modify the behaviour to fit the more complex robot. Complex robots also take more time to simulate.

The main problem with evolutionary robotics is that the robot doesn't have a memory, so it needs to receive information from the environment. The environment is often artificially modified so that the robot can evolve into something interesting.

It is difficult to simulate the output of the sensors (sensors are usually not linear with respect to their inputs). Lookup tables or Gaussian noise are methods used to increase the results of the simulation.

Minimal simulation: the idea is to simulate only the necessary characteristics, and other parts of the robot are simplified and randomized. The problem is to determine the relevant features.

Evolution vs Learning.
The idea is to combine learning and evolution. This will allow to help and guide the evolution process while adapting to changes that occur faster than a generation. But the cost is an increased unreliability (learning wrong things).
Lamarckian evolution is about transferring the learnt knowledge from one generation to the other. It is not always a good idea, as the evolution can get stuck in local maxima.

Competitive Co-evolution

Idea: have a prey and predator model, and hope that co-evolution will improve the behaviour of both species. The problem is that evolution enters into cycles, where one generation isn't better than the previous one, it is just different. This is due to the fact that the fitness function for one species is directly influenced by the behaviour of the other species. So the fitness landscape is changing. By testing species with previous generations we can notice that predators adapt to different situations, while prey use random paths.

The change in the fitness landscape is not a disadvantage; it is the reason why co-evolution works: by having a changing landscape (the landscape is smooth at first and progresses towards something rough), it is easier to get out of local maxima and find the best solution. (solves the bootstrap problem).

Some experiments (Miller & Cliff, 1997) show that having an internal parameter for the fitness (e.g.: survival time) gives better results than an external parameter (e.g.: distance between prey & predator).

Evolvable Electronics

Reconfigurable hardware (fpga) can be used in evolutionary robotics (a genotype can be mapped to the fpga configuration string).

Evolutionary Robotics can find hardware that cannot be designed with traditional methods (smaller/faster circuits but with e.g. open gates).

Constrained vs unconstrained evolution.

Extrinsic (circuit is simulated) vs Intrinsic (circuit is physically implemented).

Usually simulation works with the constrained case (combinational circuits). It is usually not possible to simulate untraditional hardware configurations.

Circuits are tuned for the evolved hardware, so they won't tolerate as much manufacturing process variations as conventional circuits. Except if the simulator simulates faults, in which case we can obtain a very fault-tolerant circuit.

Evolvable hardware is cheaper to manufacture (doesn't require processor, NN, etc…).

The most interesting example is the evolution of a low pass filter. The traditional way of building a low pass filter requires multiplications and additions. Evolvable hardware can solve the problem by using only multiplexers.

There exist analogic equivalents for the fpga (circuits with basic blocks like op-amp, resistors, capacities, etc.). The interconnection of the basic blocks can be chosen. The problem is how to avoid short-circuits !

The size of the genotype grows as the circuits become larger. This is a major problem of evolvable electronics. It is possible to reuse basic building blocks.

Developmental Systems

Idea is to use simple cell replication/differentiation to "grow" complex structures. The main problem is that it is difficult to find the simple rules that generate a given structure.

L Systems (Lindenmayer, 1968): concept of rewriting (given production rules that are applied in parallel).

Bracketed L Systems: brackets represent stack operations ([= Push,] = Pop). This is used to create trees.

Stochastic L Systems: use of probabilities (production probabilities).

Context Sensitive L Systems: another extension of L Systems, production rules are applied only if a certain symbol is preceding or following the symbol to "produce".

L Systems can be used to create neural networks.

Evolution of Morphology

Shape optimization (using GA). E.g.: wing shapes, satellite structures, table.

Architecture optimization. E.g.: Lego bridge.

Art (needs genetic encoding that allows duplication, variable length genome, etc.). Usually user (human) evaluates the design (not easy to design a fitness function).

Artificial Life (framstick, etc.).

Immune Systems (IS)

Can be applied to fight computer viruses, network intruders, electronics fault.

Human immune system is based on Lymphocytes that have antibodies. The antibodies can innate pathogens (antigens). Human IS has systems to avoid self-antigen.

There exists different types of measure for matching affinity (a match occurs if the antibody is similar to the antigen). E.g.: Euclidean, Manhattan, Hamming distances.

Negative Selection (Forrest, 1994) proposed a way to create artificial IS. The first phase is training. He generates random strings and checks (human intervention or other methods) if they should be censored.

The problem is to avoid the growth of the antibodies (growth of memory requirement). Regulatory Network solves the problem.

Detecting electronics fault: The system is trained using a system that works. The IS remembers the correct state transitions. The major problem is that a single transistor can fail the entire system, whereas in biological, the body has time to respond.

Collective and Swarm Intelligence

Key idea: use simple agents to create intelligence at the group level. It can allow the group to achieve something that the individuals cannot achieve by themselves (e.g.: creating a bridge).

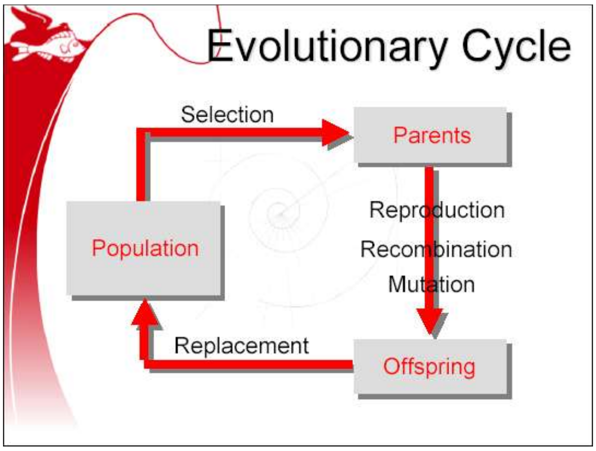
In some cases the individuals don’t know they are solving a problem.

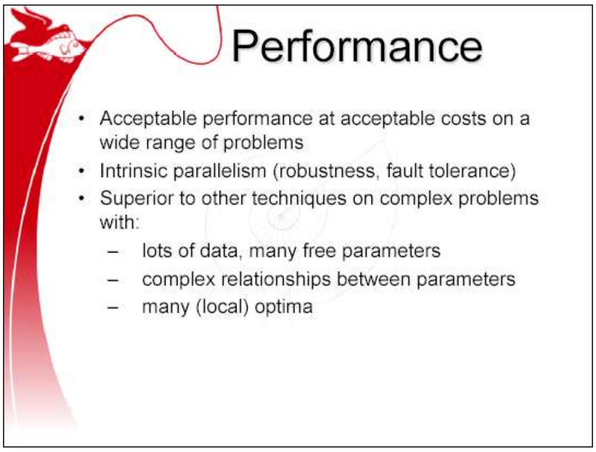
One of the problems is communication and coordination among the agents. Communication can be direct, or through the environment (stigmergy, e.g.: pheromone of ants).

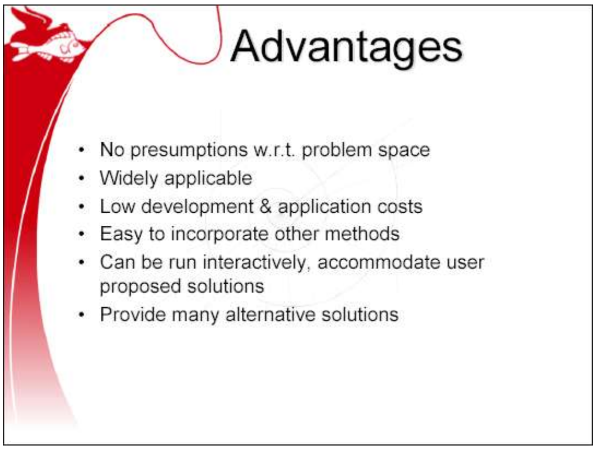
Usually there is no supervisor (distributed system) !

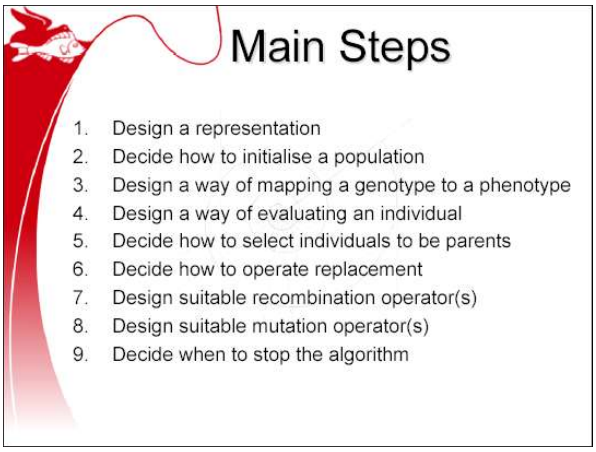
Most Important Slides

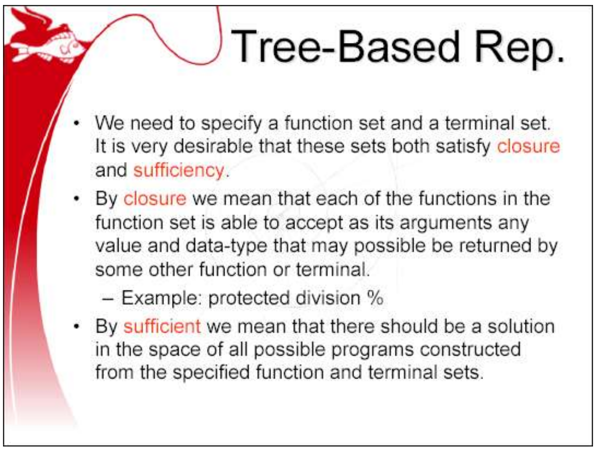
Evolutionary Systems (Genetic algorithms GA)

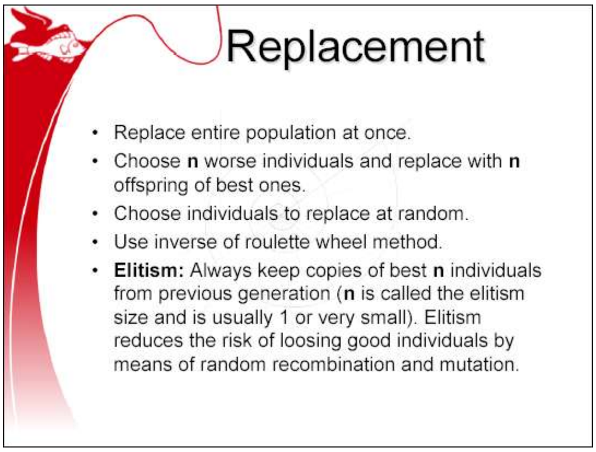


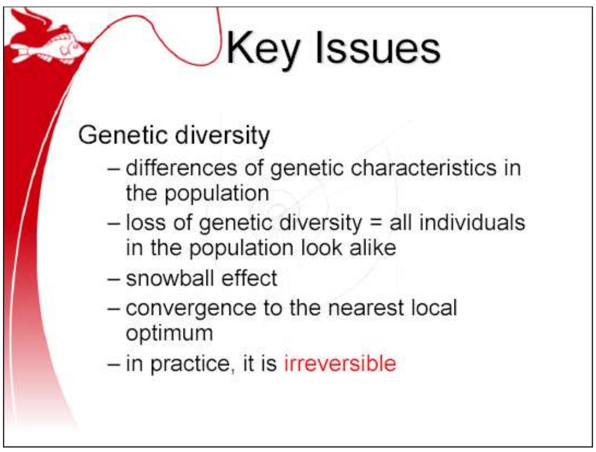


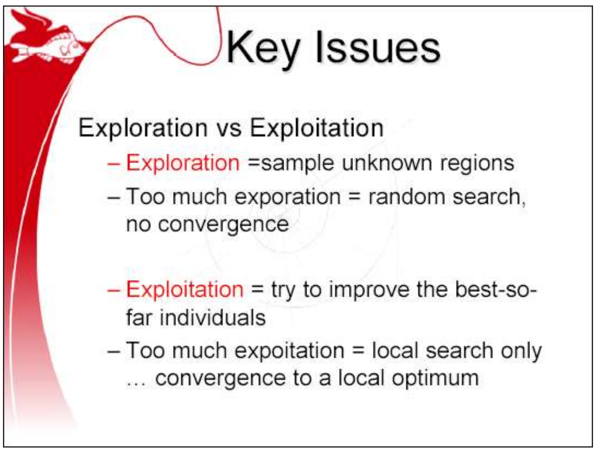








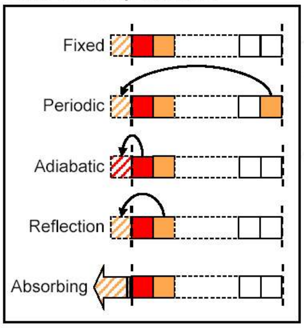




Cellular Systems and Cellular Automata (CA)

Boundary conditions

Some boundary conditions



We must specify how to build the neighborhood for the boundary cells that do not have a full neighborhood

In practice...

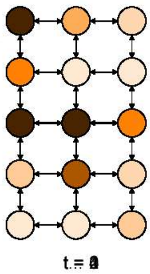
To implement and run a CA experiment

1. Assign the geometry of the CA space
2. Assign the geometry of the neighborhood
3. Define the set of states of the cells
4. Assign the transition rule
5. Assign the boundary conditions
6. Assign the initial conditions of the CA
7. Repeatedly update all the cells of the CA, until some stopping condition is met (for example, a pre-assigned number of steps is attained, or the CA is in a quiescent state, or cycles in a loop,...).

Informal definition of CA

A Cellular Automaton (CA) is

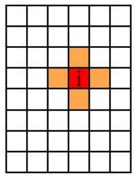
- a **geometrically structured** and
- **discrete** collection of
- **identical** (simple) systems called **cells**
- that **interact** only **locally**
- with each cell having a local **state** (memory) that can take a **finite** number of values
- and a (simple) **rule** used to **update** the state of all cells
- at **discrete time** steps
- and **synchronously** for all the cells of the automaton



Formal definition of CA

A Cellular Automaton is

- an **n-dimensional lattice** of
- identical and synchronous **finite state machines**
- whose **state** is updated following a **transition function** (or **transition rule**)
- that takes into account the state of the machines belonging to a **neighborhood** of the machine, and whose geometry is the same for all machines

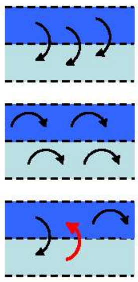


$$s_i(t+1) = \phi(s_j(t) ; j \in N_i)$$

The Growth of Complexity

von Neumann's approach:

- Usually a machine can produce only machines of lesser complexity
- If we could build a machine capable of self-reproduction we would have a machine that produces a machine of equal complexity
- If the self-reproduction process could tolerate some "error" then some of the resulting machines might have greater complexity than the original one

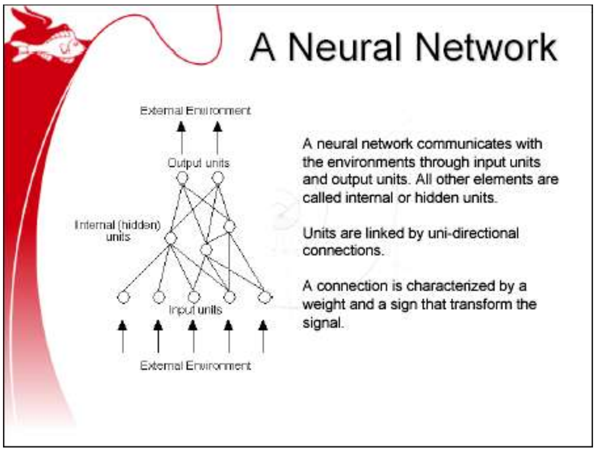


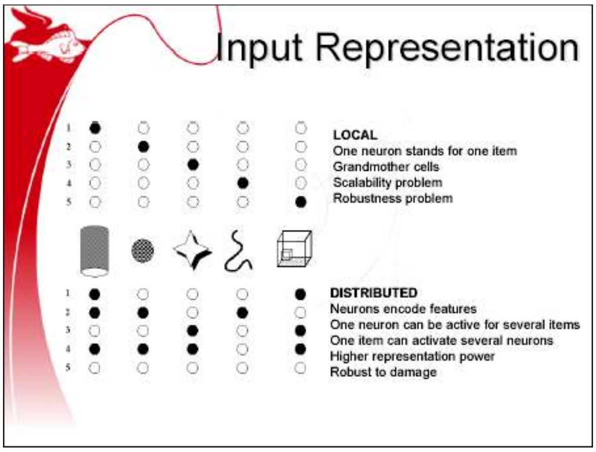
CA Summary

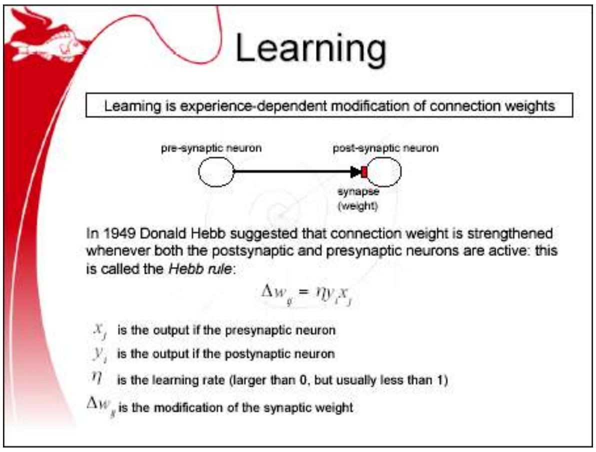
We have only scratched the surface of the CA world. However, we have seen that CA can used be at least as:

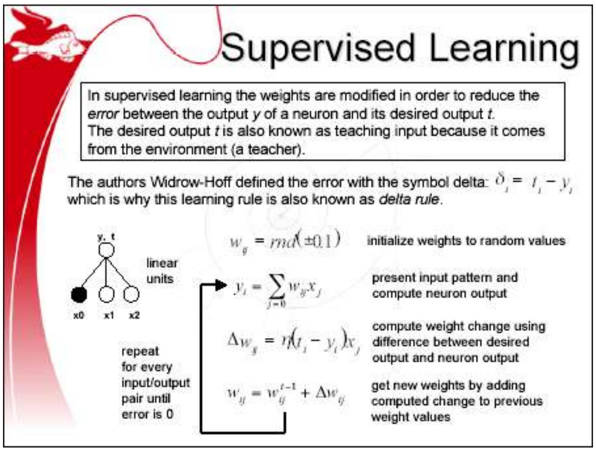
- Synthetic universes creators in Evolutionary and Artificial Life experiments.
- Models and simulators of simple and complex, biological, natural, and physical systems and phenomena.
- Computation engines.
- Testers of hypotheses about emergent physical and computational global properties and the nature of the underlying local mechanisms.

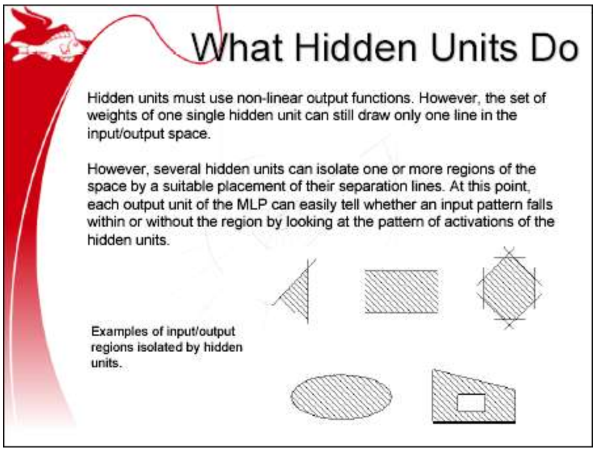
Neural Systems

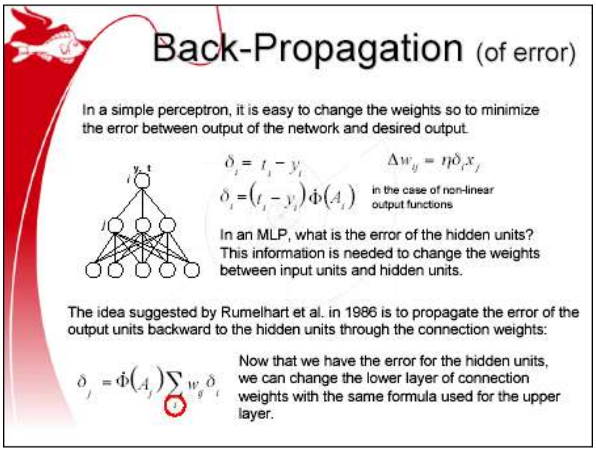




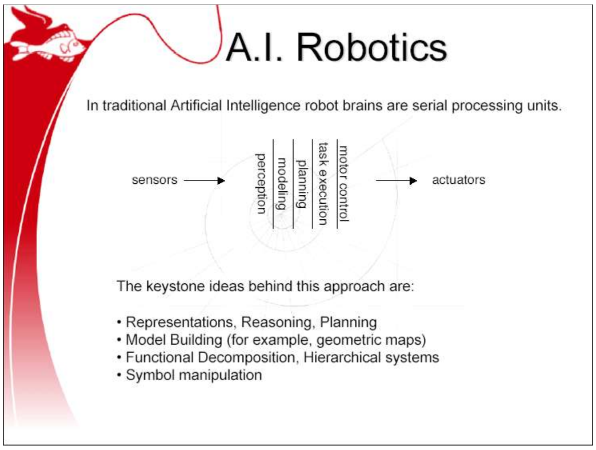


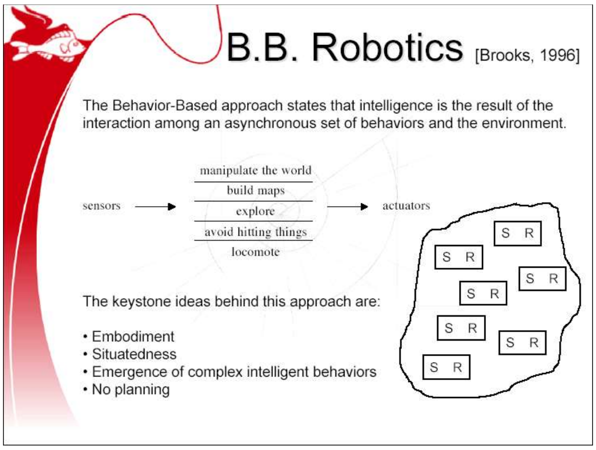


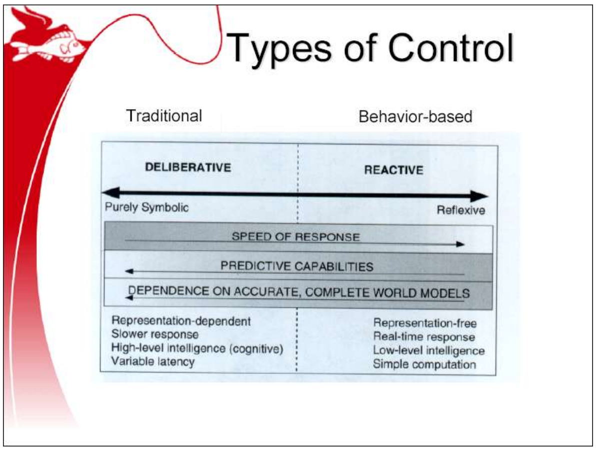


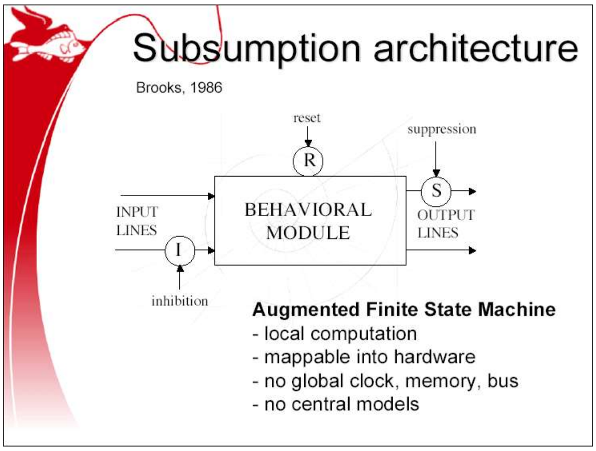


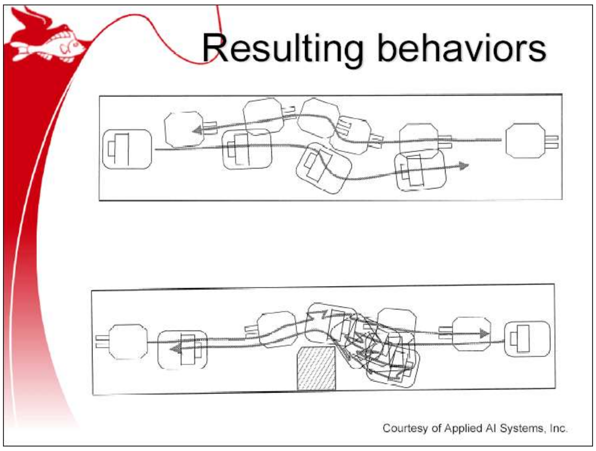
Behavioural Systems

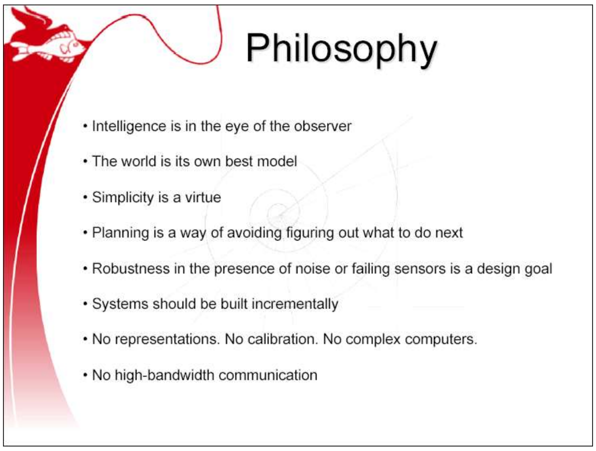




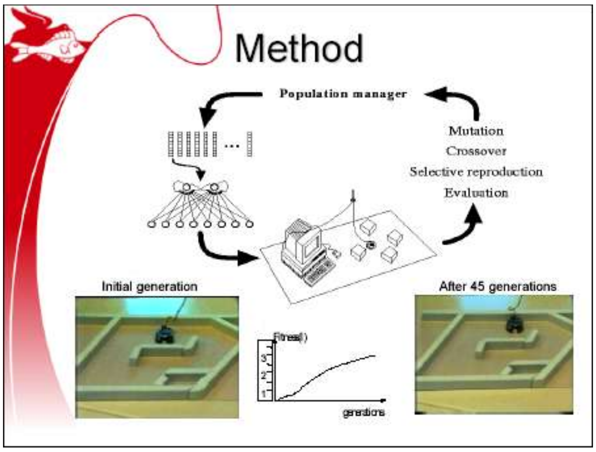


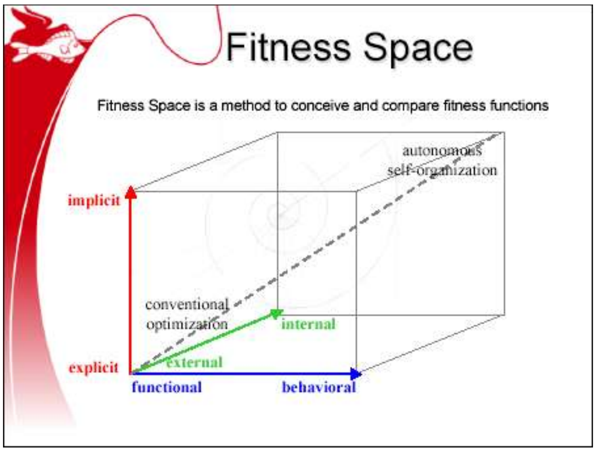


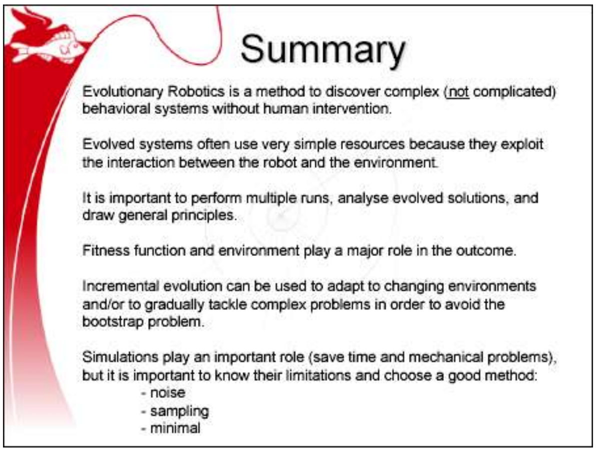


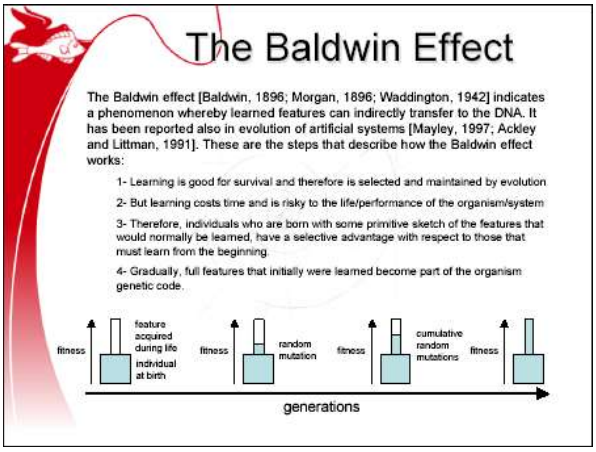


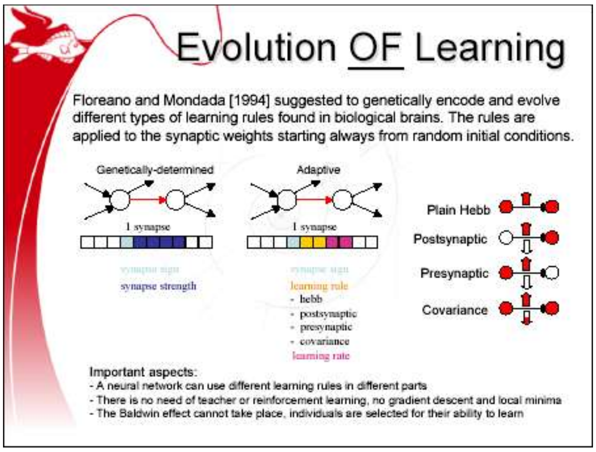
Evolutionary Robotics

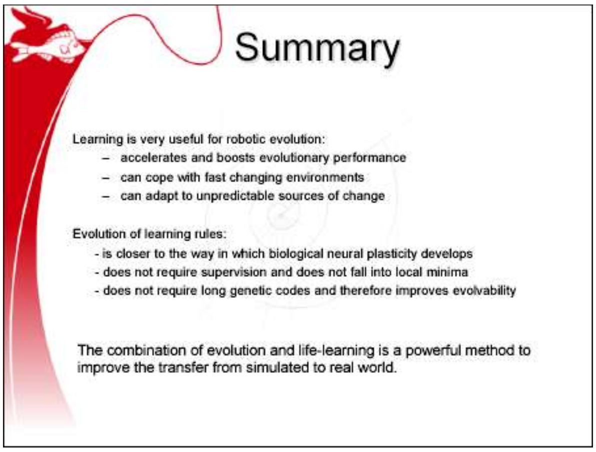




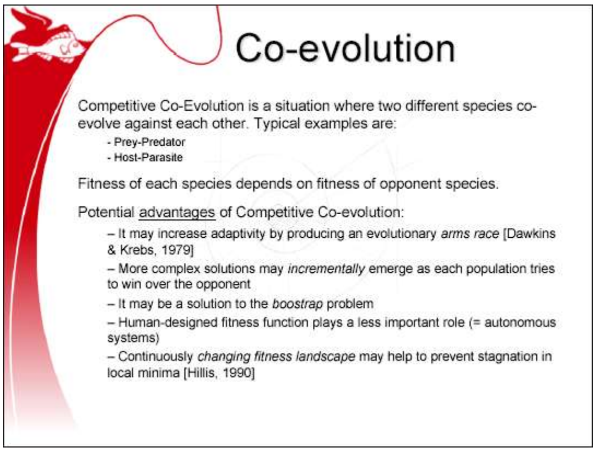


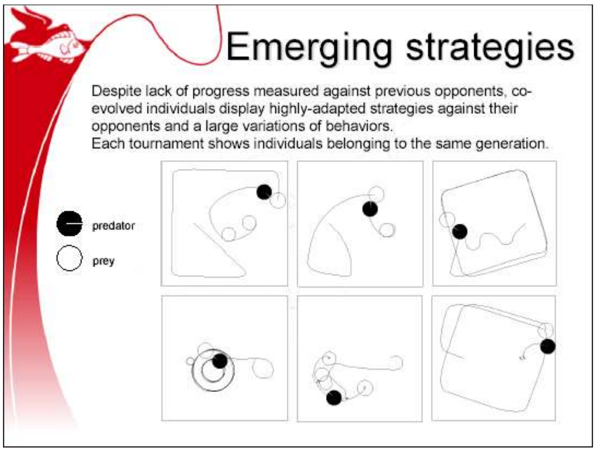


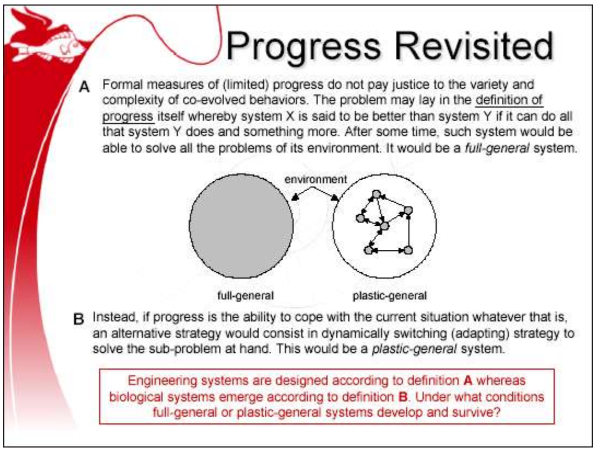


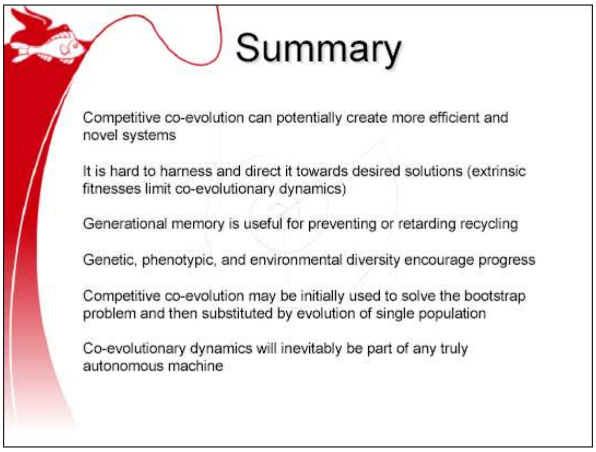


Competitive Co-evolution









Evolvable Electronics

Definitions

Unconstrained evolution: Evolution explores all the circuits
Constrained evolution: Constraints are applied to the evolved circuits

Evolution is "unconstrained" when it is free to explore all the possible circuits. Evolution may thus explore circuits that are not normally conceivable by the traditional design methodologies, possibly finding more efficient circuits, or circuits which exploit different principles than those of traditional design.

Extrinsic evolution: Circuit is simulated
Intrinsic evolution: Circuit is physically implemented

Simulations can be used without problem when they model exactly the hardware. This is generally the case with constrained circuits (e.g. simple combinational circuits).
However, when circuits have complex dynamics (e.g. unconstrained evolution), simulations cannot model exactly the physical device. Hence, extrinsic evolution often generates circuits that do not work in real hardware: circuits tune to the simulation model.

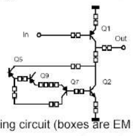
The most interesting use of EHW is unconstrained intrinsic evolution: EA can explore new circuits and test them according to the physics law, without being hampered by simulation models.

Design and evolution

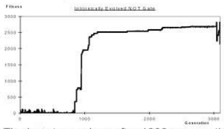
Design	Evolution
The design process is a top down approach	The evolutionary process is a bottom up approach
Key characteristics:	Key characteristics:
• The complete specification of the input-output relationships must be known (e.g. timing diagrams, truth tables)	• No need for a detailed specification: the global quality (fitness) of the circuit is used
• Hierarchical decomposition in simpler blocks	• Flattened approach (no redundancy between building blocks)
• Circuits designed to tolerate manufacturing process variations	• Circuits are often tuned to the hardware they were evolved on
• Use of mathematical models and design methodologies	• Evolution does not use mathematical models or design methodologies
• Inner working is chosen by the designer	• Inner working is chosen by the evolutionary process

Evolvable Motherboard: example

Evolution of an inverter with 10 bipolar transistors plugged in the EM (5 are used by evolution). Uncommon circuits have evolved (floating pins) [Layzell, 98]



Resulting circuit (boxes are EM switches)


The inverter evolves after 1000+ generations

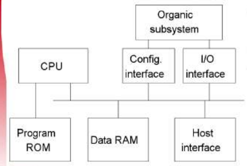
Characteristics of the EM: more flexibility!

- Independence of commercial FPGAs/FPAAs
- Discrete components may be selected according to the application
- No need to wire up the circuit by hand
- Fine- or coarse-grained evolution
- Programmable switches have a small resistance: no short-circuits

POEtic chip (3 year European project)

Generic platform for bio-inspired applications, notably multi-cellular, evolvable (P), developing (O), learning (E) circuits

Phylogenesis (artificial evolution)	Ontogenesis (development)	Epigenesis (learning)
•Evolution of circuit configuration	• Self-test and self-repair • Growth	• Adaptation in changing environments



CPU
32 bit, 64 instructions RISC CPU
• I/O, organic subsystem configuration
• Evolution

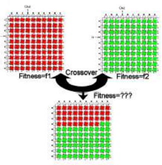
Organic subsystem
Reconfigurable hardware
• Multi-cellular system
• Development, learning

Challenges

Why don't we see EHW in our everyday life?
Well, many challenges have still to be solved!

Evolution of circuits is difficult (i.e. need a lot of generations), because of two factors:

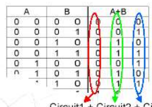
- Scalability: the size of the genotype generally grows with the size of the circuit (for each element some bits are needed) and generate a large search space which is difficult for EA to work on.
- Evolvability: the application of the genetic operators on the chromosome do not generate smooth fitness landscapes.



Possible solutions

- Incremental evolution

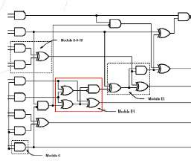
Solve subsets of the problem: simpler circuits are evolved and combined later on. Only possible if subproblems can be identified.



Circuit1 + Circuit2 + Circuit3 = Complete circuit

- Reuse building blocks

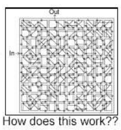
Instead of evolving a circuit from basic elements, higher level blocks are detected in the circuit and used as such. Difficulty: how to detect such blocks?



More challenges

Some challenges typical of unconstrained evolution:

- Operational envelope is unknown: does a circuit evolved on one chip work on another chip?
  Test the circuit on many chips during evolution
- Inner working may not be understandable
  Circuit may still serve as a building block



How does this work??

- Simulation and hardware implementation do not match
  Test the circuit in simulation *and* in hardware

Conclusion

Key points of EHW:

- No need for a complete circuit specification
- No need to define how the circuit works
- Achieve better use of hardware resource
- Discover new uncommon and more efficient circuits
- EHW has the properties of evolved systems: adaptability, fault tolerance, etc.

Developmental Systems

Overview

Developmental systems attempt to capture mechanisms of growth of biological systems. In nature, growth is given by a process of cell duplication and differentiation. In artificial systems, it is often based on a process of iterated symbol rewriting.

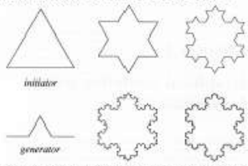
Advantages of development in artificial systems:
- Complex structures can be described by few symbols and rules
- Order and symmetry emerge naturally
- Development provides adaptation when interacts with environmental context

Disadvantages of development in artificial systems:
- May generate complex structures to solve simple problems
- Difficult to introduce irregularities if necessary
- Difficult to determine symbols and rules necessary to generate a desired structure (-> use evolution)

Rewriting systems

Rewriting is a technique for defining complex objects by successively replacing parts of a simple initial object using a set of rewriting rules, or *production rules.*

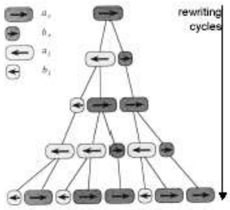
Snowflake curves can be generated by replacing the edges of a polygon with open polygons [von Koch, 1905]. At each iteration, the open polygon is shrinked to fit the lengths of the polygon edges.



Several other types of rewriting systems have been developed. These include variations of cellular automata (Conway's game of life) and language systems [Chomsky, 1956].

Definition

L-systems are rewriting systems that operate on character strings. An L-system is composed of a set of characters over an alphabet, an axiom ω (start structure), and a set of production rules *p*. Production rules are applied in parallel and replace all characters in the string.



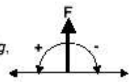
Example:
Development of blue-green bacterium *Anabaena catenula:*
ω : a_r
p_1 : a_r -> $a_l b_r$
p_2 : a_l -> $b_l a_r$
p_3 : b_r -> a_r
p_4 : b_l -> a_l

If no production rule is specified for a character s, then we assume the identity production rule p : s -> s

Turtle Interpretation

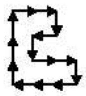
Prusinkiewicz gave L-systems a graphic interpretation based on LOGO-style turtle geometry.

The state of the turtle is defined as a triplet (x, y, α) where the Cartesian coordinates (x, y) represent the turtle's *position* and the angle α, also known as *heading*, represents the direction in which the turtle is facing.



Given the step size *d* and the angle increment δ, the turtle can respond to the following commands:

F move forward a step of length *d* while drawing a line. New state of the turtle is (x', y', α) where $x'=x + d\cos\alpha$ and $y'=y' + d\sin\alpha$.
f as above, but do not draw the line.
\+ turn left (counterclockwise) by angle δ. The new state of the turtle is (x', y', $\alpha+\delta$).
\- turn right (clockwise) by angle δ. The new state of the turtle is (x', y', $\alpha-\delta$).

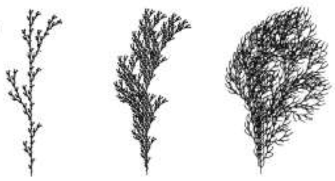


δ = 90
FFF-FF-F-F+F+FF-F-FFF

Bracketed L-systems

Two new symbols are required to build trees so that after a branch has been constructed, the turtle goes back to its original state and continues the construction of the tree.

[push current state of the turtle (position, orientation, color, thickness, etc.) onto a pushdown stack.
] pop a state from the stack and make it the current state of the turtle (position changes, but no line is drawn).
δ = 45°
F[+F][-F[-F]F]F[+F][-F]



a
n=5, δ=25.7°
F
F → F[+F]F[-F]F

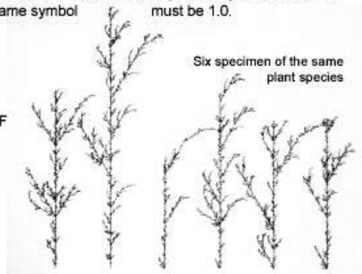
b
n=5, δ=20°
F
F → F[+F]F[-F][F]

c
n=4, δ=22.5°
F
F → FF-[-F+F+F]+[+F-F-F]

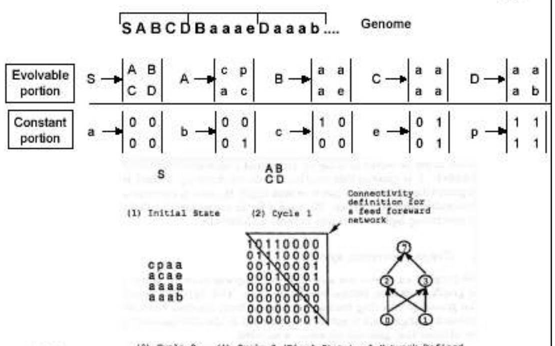
Stochastic L-systems

All plants generated by the same L-system are identical, but in nature there are no two equal specimen. Specimen-to-specimen variation can be modeled by introducing *production probabilities.* For every symbol, there is one or more production rules with an associated probability. The sum of all probabilities over the same symbol must be 1.0.

ω : F
p_1 : F $\xrightarrow{.33}$ F[+F]F[-F]F
p_2 : F $\xrightarrow{.33}$ F[+F]F
p_3 : F $\xrightarrow{.34}$ F[-F]F



Grammar Encoding



[Kitano, 1990]

Summary

L-systems are interesting for computer graphics and exploring space of topological relationships in developing biological systems.

They represented a major step into modeling and understanding plants.

It is difficult to find the set of rules and symbols behind the generation of complex structure with desired characteristics.

Two methods are used:
- Exhaustive exploration within a small space of alphabets and production rules
- Artificial evolution

Architectures of neural networks can be developed using rewriting strategies. In addition, development can take place during robot operation and benefit from environmental context by adapting the final architecture.

Still largely unexplored field. Interactions between evolution, morphogenesis and learning have not yet been studied!

Evolution of Morphology

Creativity & Computers

Design is an art of creation.
Can computers be creative and how?

1. Creativity is only possible by going beyond the bounds of a representation and by finding a novel solution which could not have been defined by that representation (computers cannot be creative) [Boden, 1992]
2. A computer is creative when it explores the possible design state spaces in addition to exploring the parameters within individual design spaces (computers can be creative) [Gero, 1996]
3. The lesser the knowledge about existing relationships between the requirements and the form to satisfy those requirements, the more a design problem tends towards creative design (computers can be creative) [Rosenman, 1997]

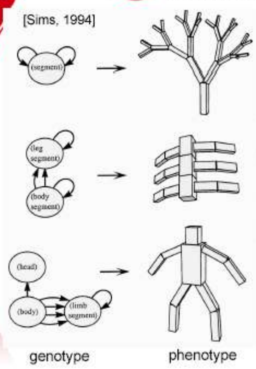
Artificial Life

Artificial Life is evolution of morphology and brain of artificial creatures that live in environments. Evolutionary Robotics is a form of Artificial Life.

1. Define building blocks and developmental rules for body and brain
2. Develop genetic encoding that allow duplication, variable-length genomes, redundancy, redundancy, gene expression, etc.
3. Design implicit fitness functions (energy, movement, etc.)
4. Criteria for success: survival of creatures

Virtual Creatures

[Sims, 1994]



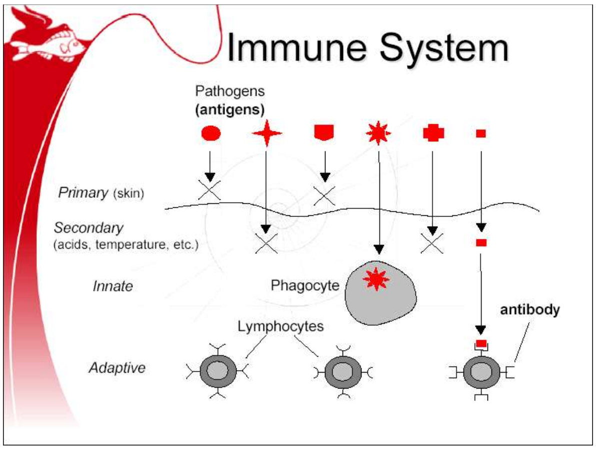
Body representation is directed graph. Nodes have properties:
- dimension
- joint type (rigid, twist, revolute, ...)
- recursive-limit
- connection (position, orientation, scale, reflection)
- terminal
- neural circuit

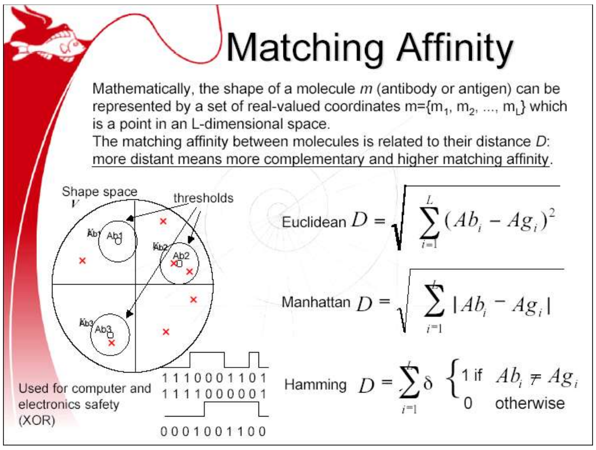
Neural circuit representation is directed graph. Nodes have properties:
- sensor
  - joint sensor
  - contact sensor
  - photosensor
- neuron (math type)
  - sum
  - memory
  - oscillator
  - max, etc.
- effector (force on muscle)
  - positive/negative (push/pull)

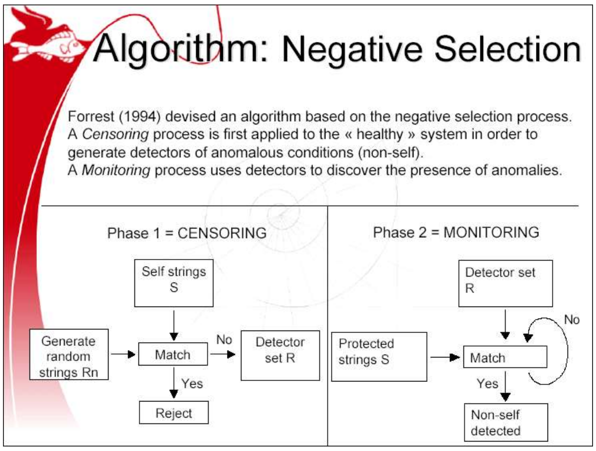
Summary

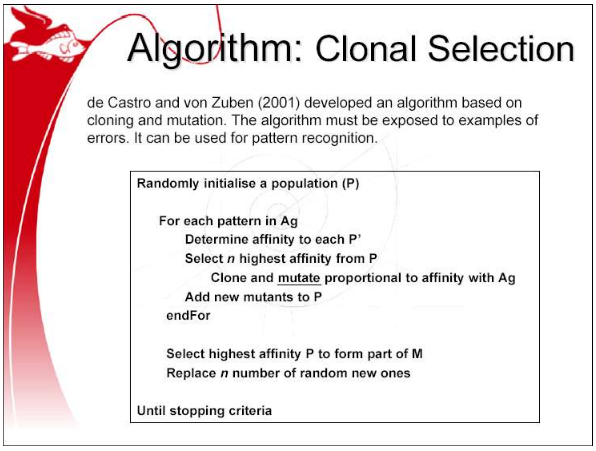
- Artificial Evolution can discover innovative and efficient morphologies
- Fitness evaluation is crucial for realism and applicability
- For optimization problems, must comply with industry standard software
- Computation speed is major problem:
  - Start with finite element methods, then move on to realistic simulator
  - Combine simulation with real implementations
  - Use neural networks to predict hardware behavior and mix with simulation

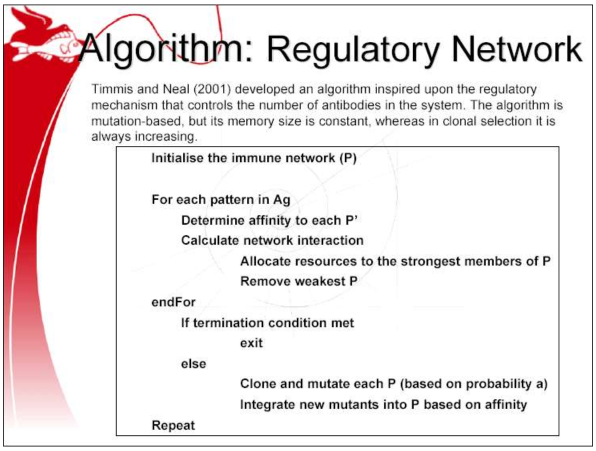
Immune Systems (IS)

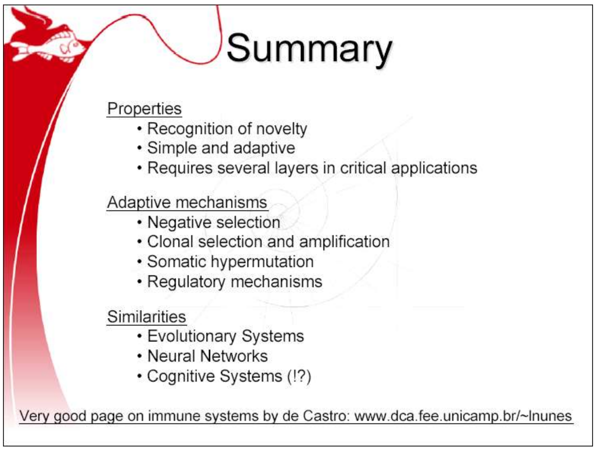












Collective and Swarm Intelligence