Natural Computing

Lecture 2: Genetic Algorithms

School of Informatics

J. Michael Herrmann
michael.herrman@ed.ac.uk
phone: 0131 6 517177
Informatics Forum 1.42

INFR09038

23/9/2011

Meta-heuristic algorithms

- Similar to stochastic optimization
- Iteratively trying to improve a possibly large set of candidate solutions
- Few or no assumptions about the problem (need to know what is a good solution)
- Usually finds good rather than optimal solutions
- Adaptable by a number of adjustable parameters

http://en.wikipedia.org/wiki/Metaheuristic

1. Chapter

Genetic algorithms

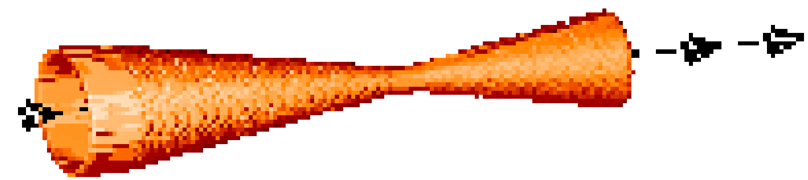
An early example of an evolutionary algorithm

Experimental contour optimization of a supersonic flashing flow nozzle

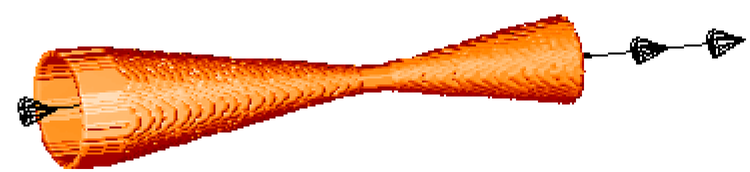
(1967-1969)

Hans-Paul Schwefel

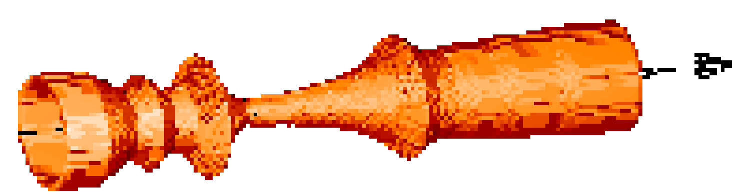
Start



Evolution



Result



More recent work: “List of genetic algorithm applications” at wikipedia.org

Paralipomena

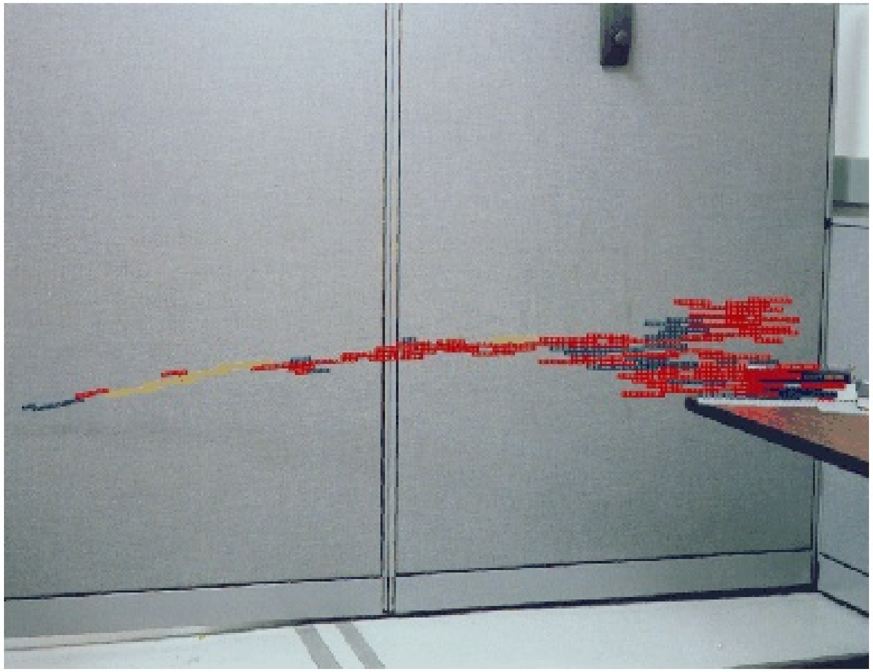
- Theory of natural evolution
- Genetics, genomics, bioinformatics
- *The Philosophy of Chance* (Stanislaw Lem, *1968*)
- Memetics (R. Dawkins: *The Selfish Gene*, 1976)
- Neural Darwinism -- *The Theory of Neuronal Group Selection* (Gerald Edelman, 1975, 1989)
- (artificial) Immune systems
- Evolution of individual learning abilities, local heuristics
- Computational finance, markets, agents

Genetic Algorithms

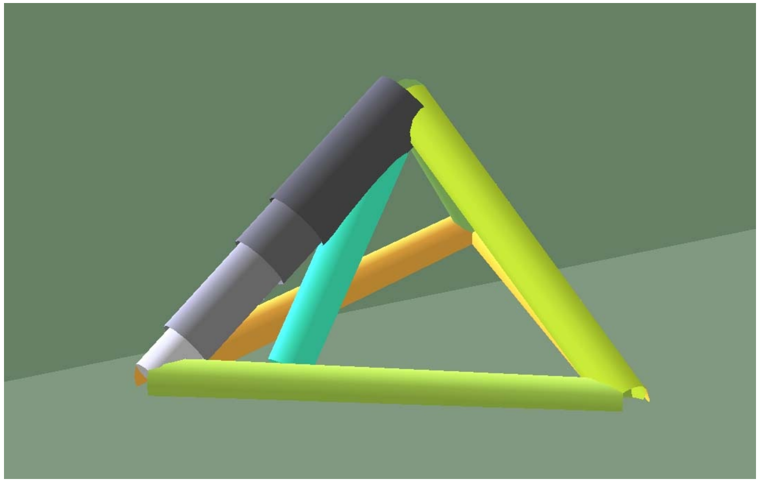
- global search heuristics
- technique used in computing
- find exact or approximate solutions to optimization problems

Applications in

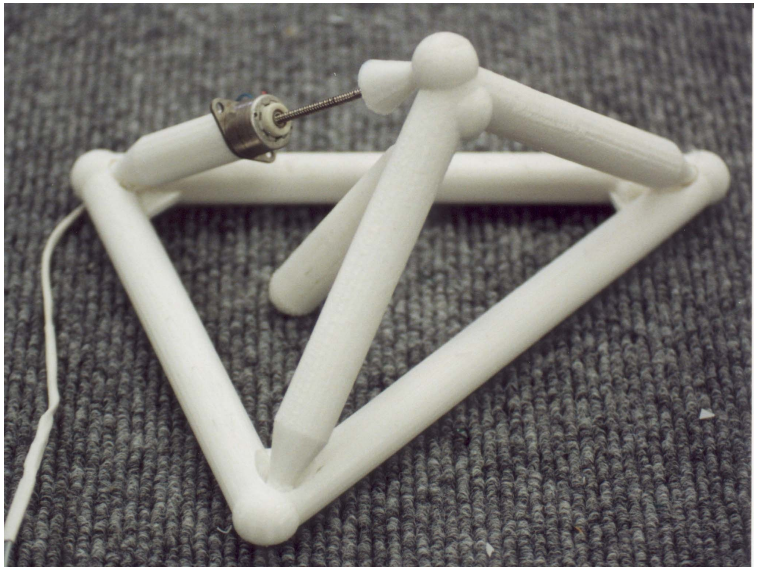
- Bioinformatics
- Phylogenetics
- Computational science
- Engineering
- Robotics
- Economics
- Chemistry
- Manufacturing
- Mathematics
- Physics

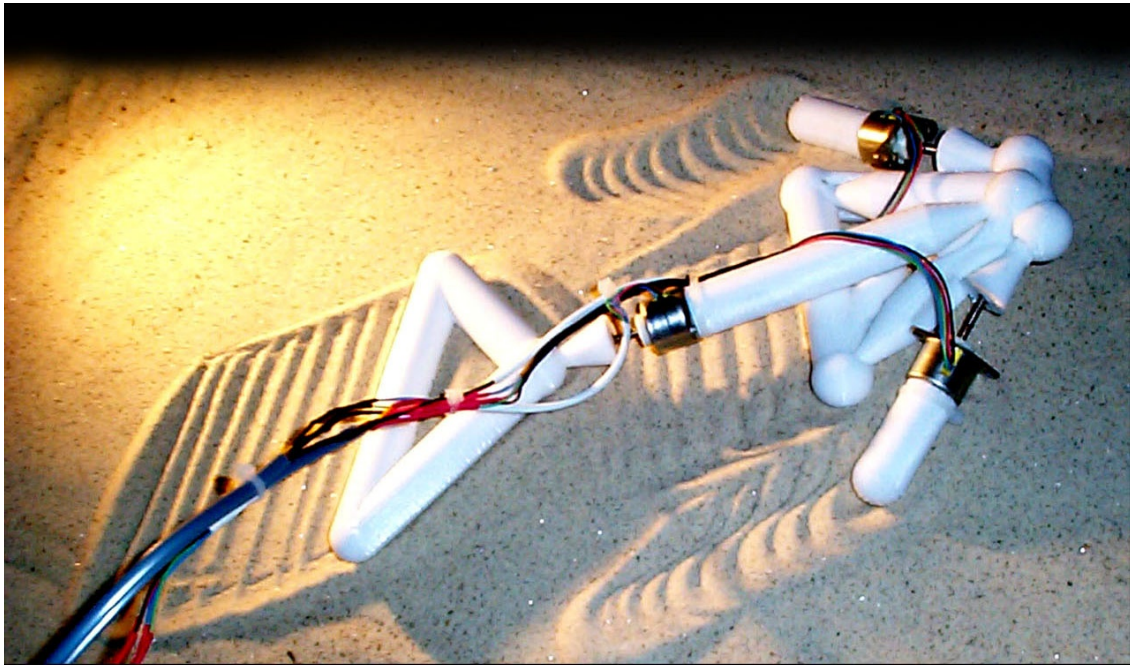


The Golem Project

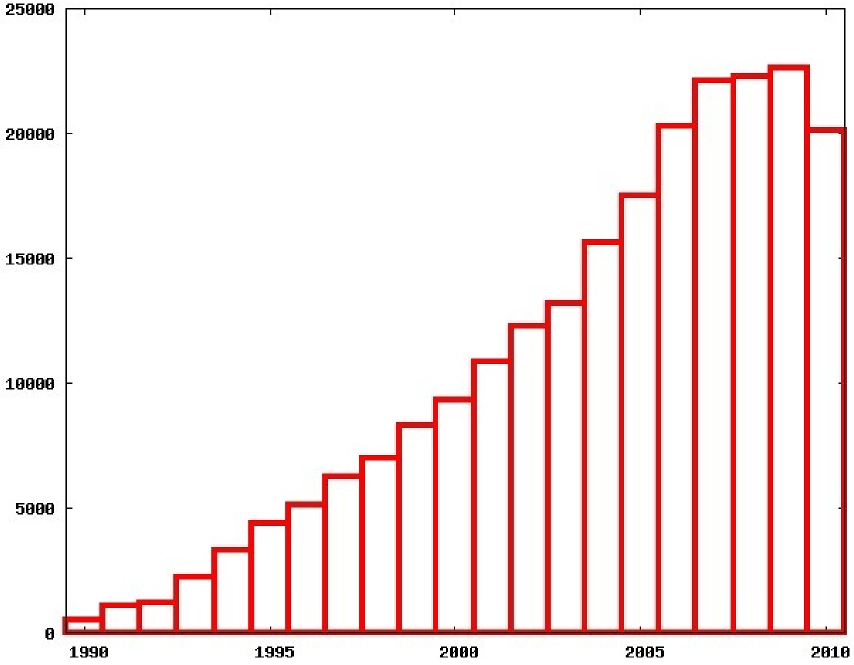


Hod Lipson &
Jordan B. Pollack (2000)

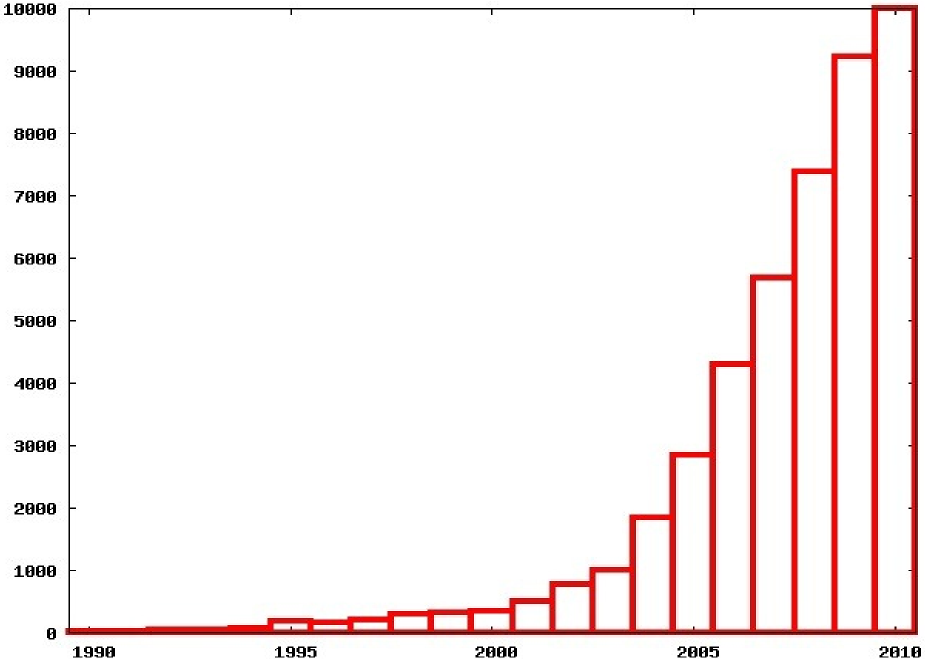




Recent scientific activity in MHA



"Genetic algorithms"



"Particle swarms"

Source: Google scholar

A Simple Example

Optimal assignment problem (OAP)

Consider the Tutor Allocation Problem

Jobs: $\text{Job}_1, \text{Job}_2, \ldots, \text{Job}_m$

Job_i is a single tutorial to be taught:

- subject, e.g. Java, IVR
- slot, e.g. Wednesday 4:10 - 5pm
- place, e.g. 4.07 Appleton Tower
- knowledge, skills required, e.g. strong at Java, some knowledge of AI techniques useful

A Simple Example

One tutor teaches

each tutorial.

We have a pool of

tutors to choose from:

Tutors:

Tutor A, Tutor B, Tutor C, …

Properties of tutors:

- knowledge/skills
- cost per hour
- time preferences
- room preferences
- optimal number of jobs

Solutions

A **solution** is an allocation of tutors to jobs:

Job:	1	2	3	4	5	6	7	8	9	10
Tutor:	A	B	C	D	E	F	G	H	I	J

Each job-tutor pairing can be given a **score**, based on how good the knowledge/skills match is:

Tutor A: some C++, strong at AI
Job 1: strong Java, some AI useful
– a reasonable match, though not perfect

A function $f_s(\mathrm{job}, \mathrm{tutor})$ calculates a numerical score for us for any pairing.

The **whole** solution can be given a score, based on:

– scores for job-tutor pairings

– total cost of solution

– hard constraints

– tutor preferences

The total score will be calculated from the scores for the individual parts.

The **problem** is to find the solution with the **best** score.

Possible Methods

Use exhaustive search?

– 5 tutors, 10 jobs $= 9.8 * 10^6$ solutions

– 10 tutors, 20 jobs $= 1.0 * 10^{20}$ solutions

– 15 tutors, 30 jobs $= 1.92 * 10^{35}$ solutions

– . . .

Possible Methods

Use greedy search?

Job 1 – find best tutor

Job 2 – find best tutor to give best combined score with the choice for Job 1

Job 3 – etc.

Almost certain to be sub-optimal since it commits to choices too early.

Possible Methods

Use Hillclimbing Local Search?

	1	2	3	4	5	6	7	8	9	10	Job
Solution$_i$:	A	B	E	A	B	B	$\boxed{\text{D}}$	C	E	D	Tutor

Suppose $\boxed{\text{D}}$ is the worst scorer. Try A, B, C, E

	1	2	3	4	5	6	7	8	9	10	Job
Solution$_{i+1}$:	A	B	E	A	B	B	$\boxed{\text{A}}$	C	E	D	Tutor

Continue until no improvement possible.

Prone to local maxima.

Genetic Algorithms

How about trying a biologically inspired solution based on genetics?

1. Generate a **population** of solutions:

Generation$_i$:

Solution1:	A	B	C	A	B	C	D	D	E	E
Solution2:	B	C	E	A	B	D	E	C	A	D
$\vdots$										
Solution$n$:	E	D	A	C	C	D	A	D	B	A

2. Give each solution a score, called a **fitness**.

3. Create a **new generation** of solutions by:
(a) selecting fit solutions
(b) breeding new solutions from old ones and add to generation$_{i+1}$.

4. When a sufficiently good solution has been found, stop.

A Simple Genetic Algorithm

- Selection (out of *n* solutions, greedy type):
  - Calculate $\Sigma_i f_S(\text{Job}_i, \text{Tutor}_i)$ for each solution *S*
  - Rank solutions
  - Choose the *k* best scorers ($1 \leq k \leq n$)
- Breeding (Mixing good solutions):
  - take a few of the good solutions as parents
  - cut in halves, cross, and re-glue (see next slide)
- Mutation:
  - generate copies of the mixed solutions with very few modifications
  - e.g. for $k=n/2$: two "children" for each of them

Recombination and Mutation

How does breeding work?

1. Reproduction:

Copy solution$_i$ unchanged into the next generation.

2. Crossover:

Parent1: ABCABC DDEE → ABCABCDCBA

Parent2: BAEDCA DCBA → BAEDCADDEE

Exchange of genetic material to form children.

3. Mutation:

(a) change one value in a solution to a random new value:

AEBCABDDCE → AEBCABDCCE

(b) swap two values:

AEBCABDDCE → AEBDABDCCE

(c) lots of others!

Mutation is usually done after reproduction/crossover, with low probability (1%).

How Well Does This Work?

- small problems: optimal solutions
- larger problems: optimal or near optimal given enough time
- anytime behaviour
- runs on parallel machines
- adding constraints is very easy
- used in a multitude of real applications
- wide applicability to problems in search, optimisation, machine learning, automatic programming, A-life, . . .

The Main Issues

- How do I represent a solution?
- How should I rate a solution for fitness?
- How large should the population of solutions be?
- How much selection pressure should I apply?
- What form of crossover should I use?
- what form of mutation should I use?

Next lecture: The Canonical GA

DNA figure: Access Excellence Graphics Library.
Genotype–phenotype figure: Blamire's Science at a Distance.

Towards a Canonical GA

- Numerous variants of GAs in applications
- The canonical GA highlights the principles why GAs work
- Darrell Whitley (1989) The GENetic ImplemeTOR
- A heuristic fitness function is often not a good measure of any “exact fitness”: Ranking introduces a uniform scaling across the population (evaluation)
- Direct control of selective pressure (improvement)
- Efficient coverage of the search space (diversity)

see: D. Whitley: A genetic algorithm tutorial. Statistics and Computing (1994) 4, 65-85

Conventions

- old population
- selection
- intermediate population
- recombination mutation
- new population

one generation

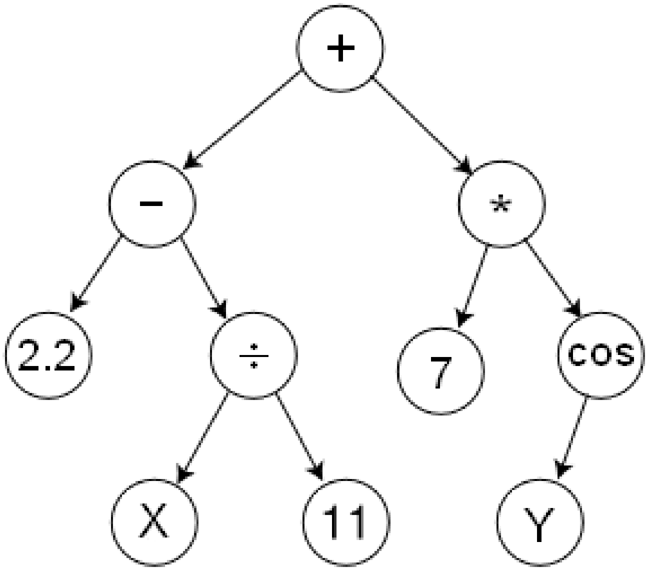
- An *individual* is a string (genotype, chromosome)
- Fitness values are replaced by ranks (high to low)
- Fitness = objective function = evaluation function

Paralipomena

- Theory of natural evolution
- Genetics, genomics, bioinformatics
- *The Philosophy of Chance* (Stanislaw Lem, *1968*)
- Memetics (R. Dawkins: *The Selfish Gene*, 1976)
- Neural Darwinism -- *The Theory of Neuronal Group Selection* (Gerald Edelman, 1975, 1989)
- (artificial) Immune systems
- Evolution of individual learning abilities, local heuristics
- Computational finance, markets, agents

Genetic Programming (GP)

- Evolutionary algorithm-based methodology inspired by biological evolution
- Finds computer programs that perform a user-defined task
- Similar to genetic algorithms (GA) where each individual is a computer program
- Optimize a population of computer programs according to a fitness landscape determined by a program's ability to perform a given computational task.



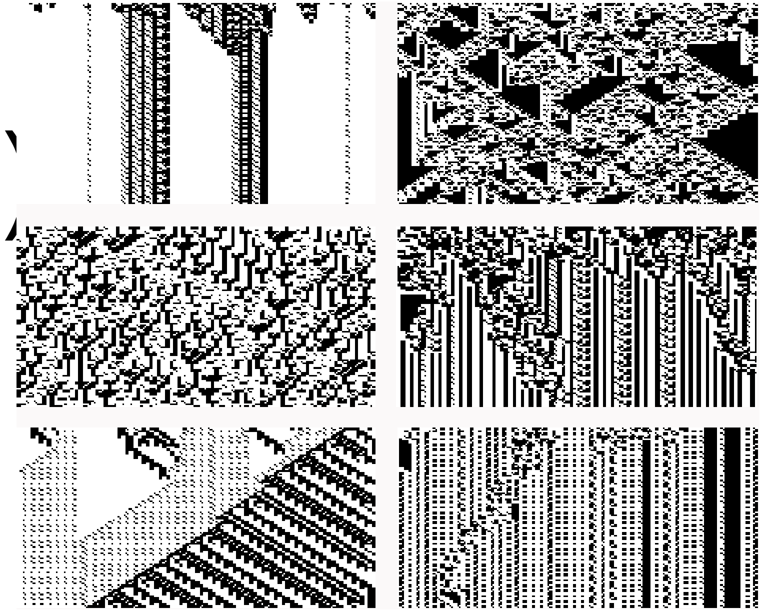
$$\left(2.2 - \left(\frac{X}{11}\right)\right) + \left(7 * \cos(Y)\right)$$

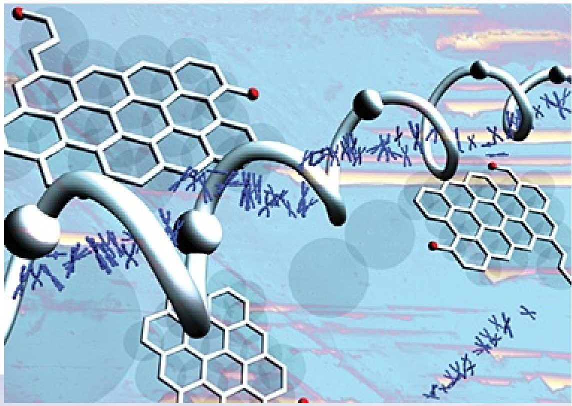
Evolutionary Computation (EC)

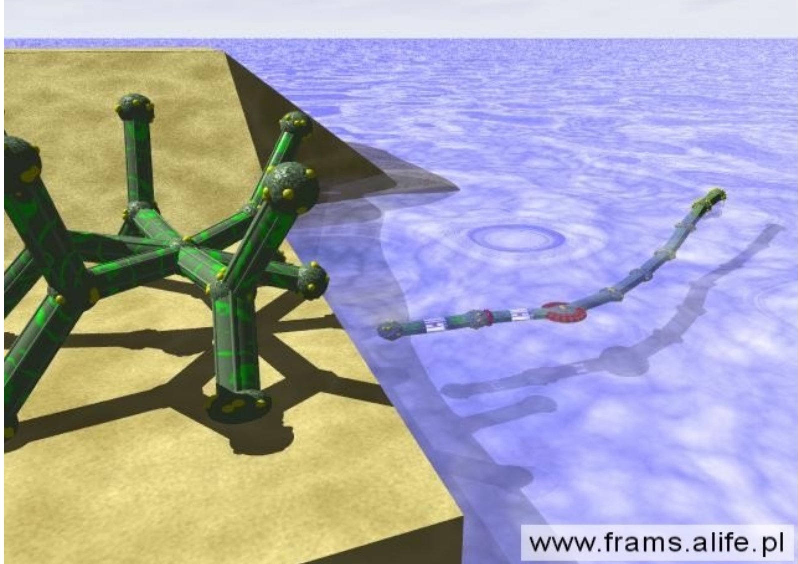
- Genetic algorithms: Solution of a problem in the form of strings of numbers using recombination and mutation
- Genetic programming: Evolution of computer programs
- Evolutionary programming: Like GP, but only the parameters evolve
- Evolution strategies: Vectors of real numbers as representations of solutions

Natural Computation (NC)

- Evolutionary Computation
- Artificial immune systems
- Neural computation
- Amorphous computing
- Ant colony optimization
- Swarm intelligence
- Harmony search
- Cellular automata
- Artificial life
- Membrane computing
- Molecular computing
- Quantum computing







Course organization

- Tuesday & Friday 15:00 – 15:50 at BS LT1
- Assignments: two assignment together worth 30% (10% + 20%) of the course mark, to be handed ir on 27 Oct / 24 Nov (both Thursdays 4pm)
- Exam: worth 70% of the course mark, taken at the end of Semester 2. Visiting students can take the exam at the end of Semester 1.
- michael.herrmann@ed.ac.uk
  phone: 0131 6 517177
  Informatics Forum 1.42
- Literature for this part:

  Melanie Mitchell: An Introduction to Genetic Algorithms. MIT Press, 1996.
  Xin-She Yang: Nature-Inspired Metaheuristic Algorithms. Luniver Press 2010

  Simulation: math.hws.edu/eck/jsdemo/jsGeneticAlgorithm.html