Evolutionary Computation

An overview

Andrea Roli

DEIS-Cesena
Alma Mater Studiorum Università di Bologna
Cesena (Italia)

andrea.roli@unibo.it

Inspiring principle

Evolutionary Computation is inspired by **natural selection**

Natural selection

"The process by which traits become more or less common in a population due to consistent effects upon the survival or reproduction of their bearers. It is a key mechanism of evolution."

(From Wikipedia, visited on December 2010)

Key concepts

- The *fittest* individuals have a high chance of having a numerous offspring.
- The children are similar, but not equal, to the parents.
- The traits characterizing the fittest individuals spread across the population, generation by generation.

EC techniques are not meant to simulate the biological evolutionary processes, but rather aimed at exploiting these key concepts for problem solving.

Evolutionary Computation

Evolutionary Computation encompasses:

- Genetic Algorithms
- Genetic Programming
- Evolution Strategies
- . . .

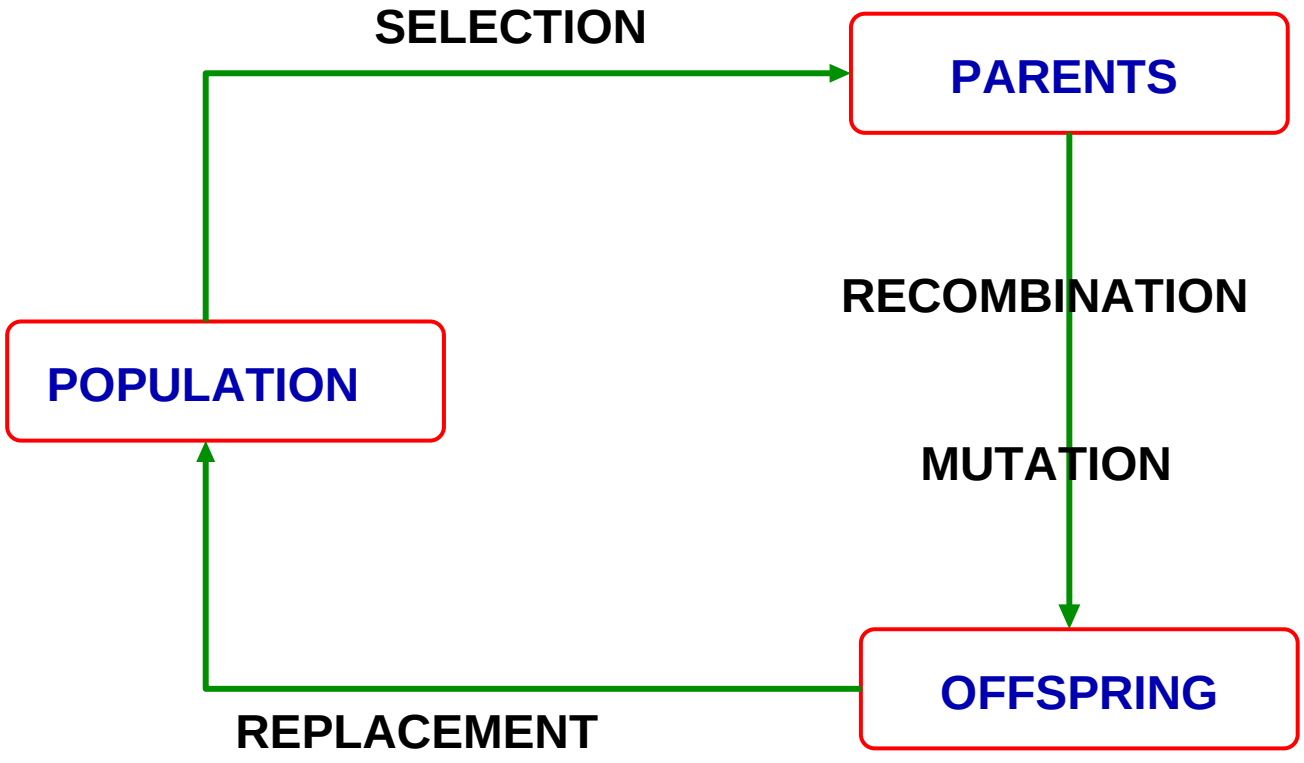
Main applications

- System design
- Neural network training
- Signal processing
- Optimization (discrete and continuous)
- Time series analysis and forecasting
- Artificial Life
- Games

The Metaphor

BIOLOGICAL EVOLUTION		ARTIFICIAL SYSTEMS
Individual	$\leftrightarrow$	A possible solution
Fitness	$\leftrightarrow$	Quality
Environment	$\leftrightarrow$	Problem

The Evolutionary Cycle



Main genetic operators

Recombination: combines the genetic material of the parents.

Mutation: introduce variability in the genotypes.

Selection: acts in the choice of parents whose genetic material is then reproduced with variations.

Replacement/insertion: defines the new population from the new and the old one.

- EC algorithms define a basic computational procedure which uses the genetic operators.
- The definition of the genetic operators specifies the actual algorithm and depends upon the problem at hand.

Genetic Algorithms

Developed by John Holland (early ’70) with the aim of:

- Understand adaptive processes of natural systems
- Design robust (software) artificial systems

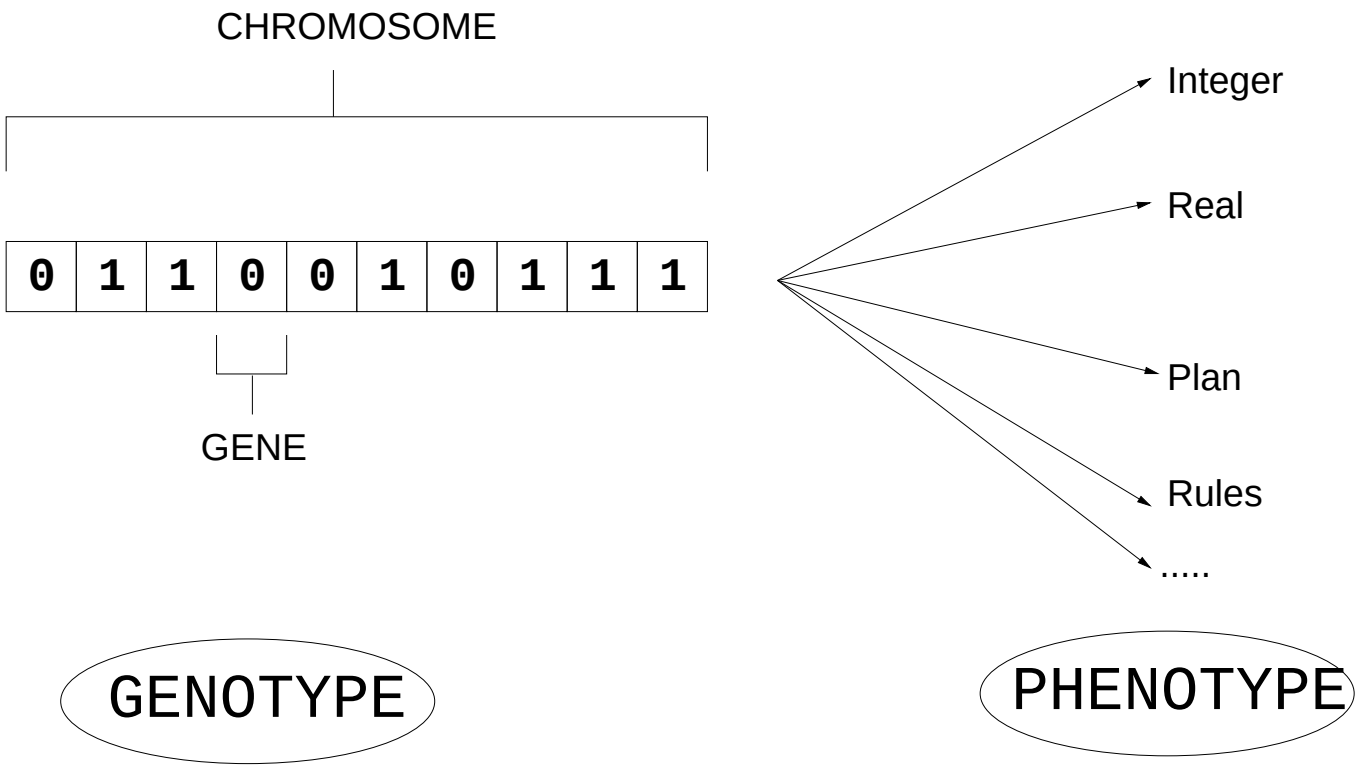
- Directly derived from the natural metaphor
- Very simple model
- ‘Programming oriented’

A bit of terminology

- A **population** is the set of individuals (solutions)
- Individuals are also called **genotypes**
- Chromosomes are made of units called **genes**
- The domain of values of a gene is composed of **alleles** (e.g., binary variable $\leftrightarrow$ gene with two alleles)

Simple Genetic Algorithm

Solutions are coded as **bit strings**



Encoding examples

Optimization of a function of integer variable $x \in [0, 100]$. Possible encodings:

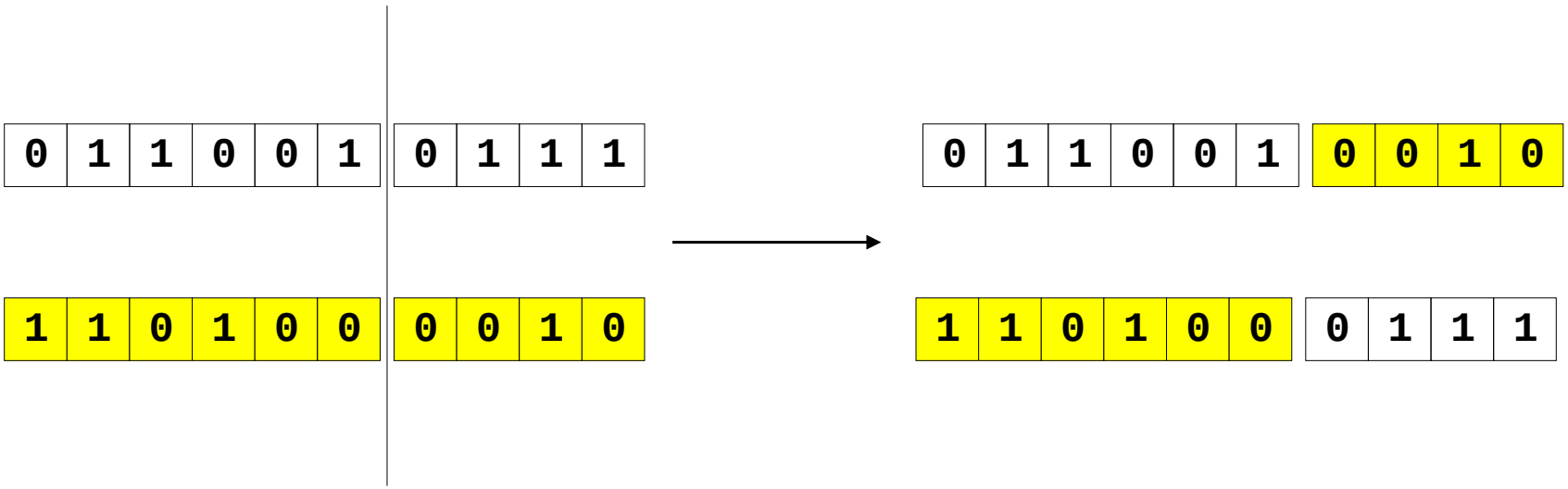
- binary coding $\longrightarrow$ string of 7 bit;
- 4 bits per digit $\longrightarrow$ string of 12 bit.

Optimization of a function of real variable $y \in [0, 1[$. Possible encoding:

- binary coding $\longrightarrow$ string

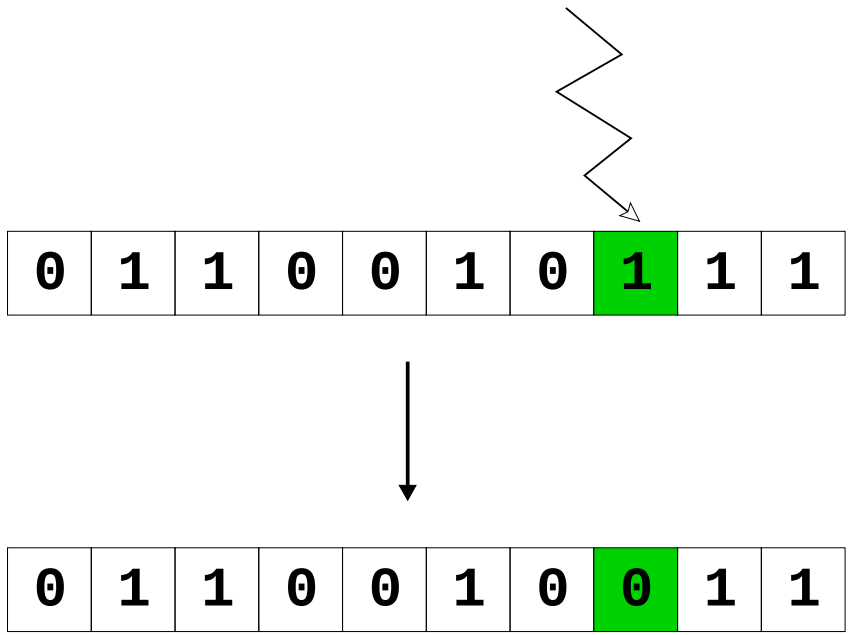
SGA genetic operators (1)

Recombination or **Crossover**: cross-combination of two chromosomes (loosely resembling biological crossover)



SGA genetic operators (2)

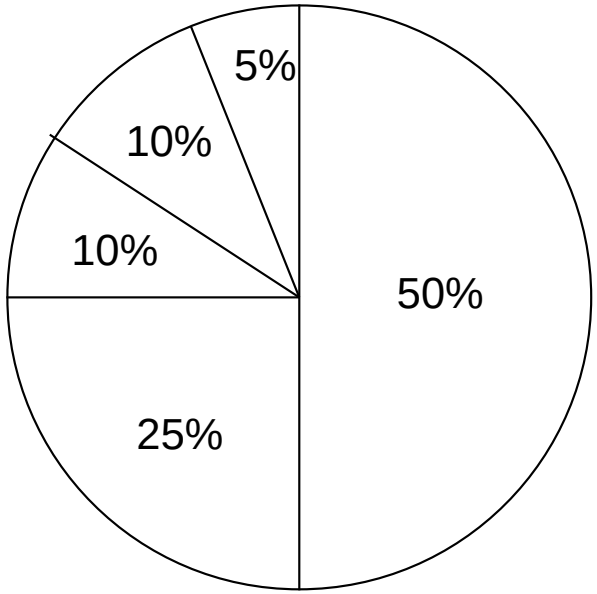
Mutation: each gene has probability p_M of being modified ('flipped')



SGA genetic operators (3)

→ **Proportional selection**: the probability for an individual to be chosen is proportional to its fitness.

Usually represented as a roulette wheel.



I_1	50
I_2	25
I_3	10
I_4	10
I_5	5

Genetic operators (4)

Generational replacement: The new generation replaces entirely the old one.

- Advantage: very simple, computationally not expensive, easier theoretical analysis.
- Disadvantage: it might be that good solutions are not maintained in the new population.

SGA: High-level algorithm

Initialize Population
Evaluate Population
while Termination conditions not met **do**
  while New population not completed **do**
    Select two parents for mating
    Apply crossover
    Apply mutation to each new individual
  end while
  Population $\leftarrow$ New population
  Evaluate Population
end while

Termination conditions

- Execution time limit reached.
- Satisfactory solution(s) have been obtained.
- Stagnation (limit case: the population converged to the same individual)

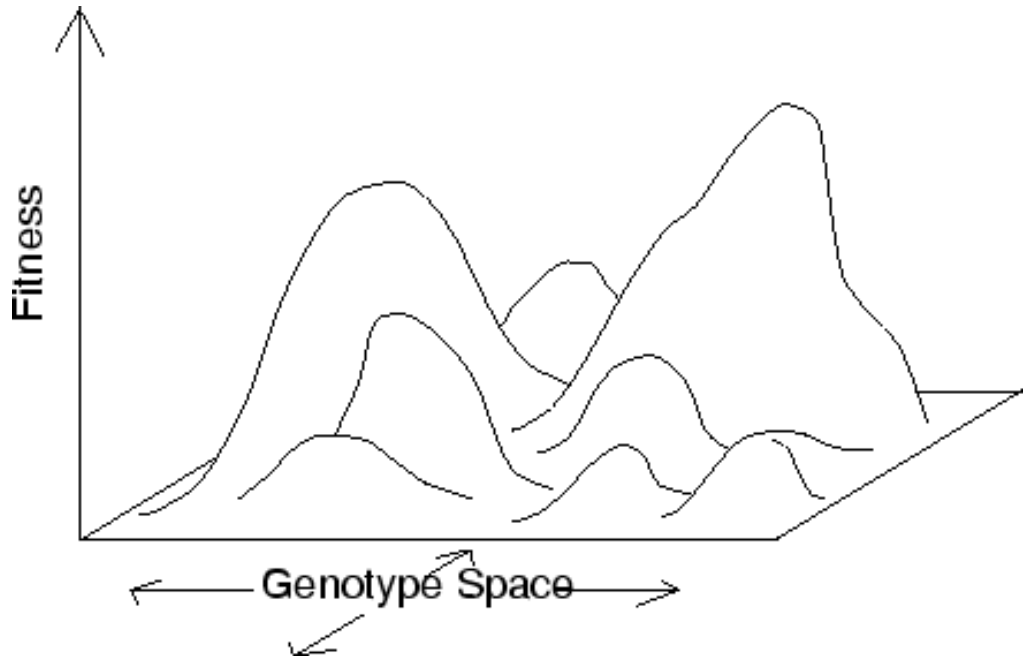
Why does it work?

Intuition:

- Crossover combines good parts from good solutions (but it might achieve the opposite effect).
- Mutation introduces diversity.
- Selection drives the population toward high fitness.

Fitness Landscape

Representation of the space of all possible genotypes, along with their fitness.



Fitness Landscape

The metaphor of landscape should be taken *cum grano salis*.

- One operator, one landscape.
- In some cases fitness landscapes are *dynamic*.
- Landscape ‘intuition’ might be misleading, because it might implicitly suggest a metric in the search space that actually does not exsist.

SGA: pros and cons

Pros:

- Extremely simple.
- General purpose.
- Tractable theoretical models.

Cons:

- Coding is crucial.
- Too simple genetic operators.

A general GA

- Solution coding (e.g., bit strings, programs, arrays of real variables, etc.)
- Define a way for evaluating solutions (e.g., objective function value, result of a program, behavior of a system, etc.)
- Define genetic operators. Mutation is always necessary, while crossover can be omitted.

Mutation

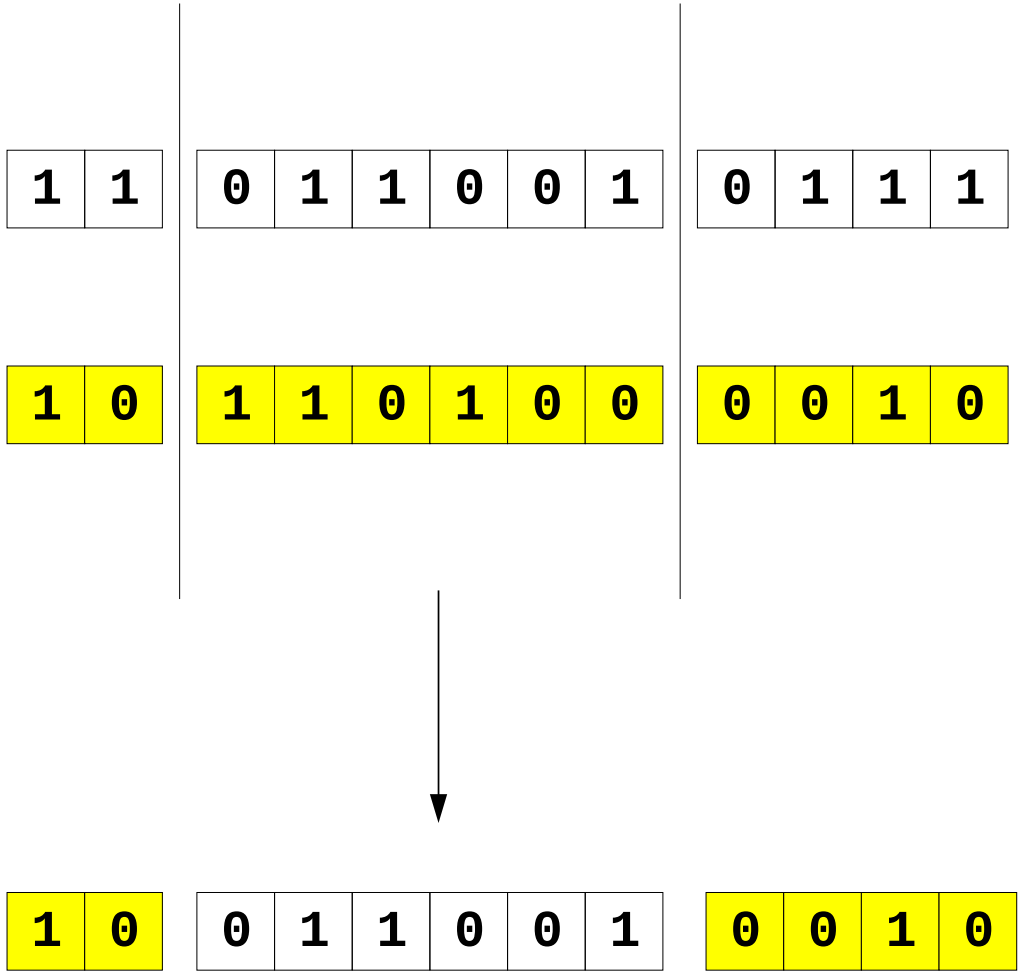
- Learning applied to modify the chromosome
- In optimization, hill-climbing or more complex local search algorithms can be applied

Examples of crossover

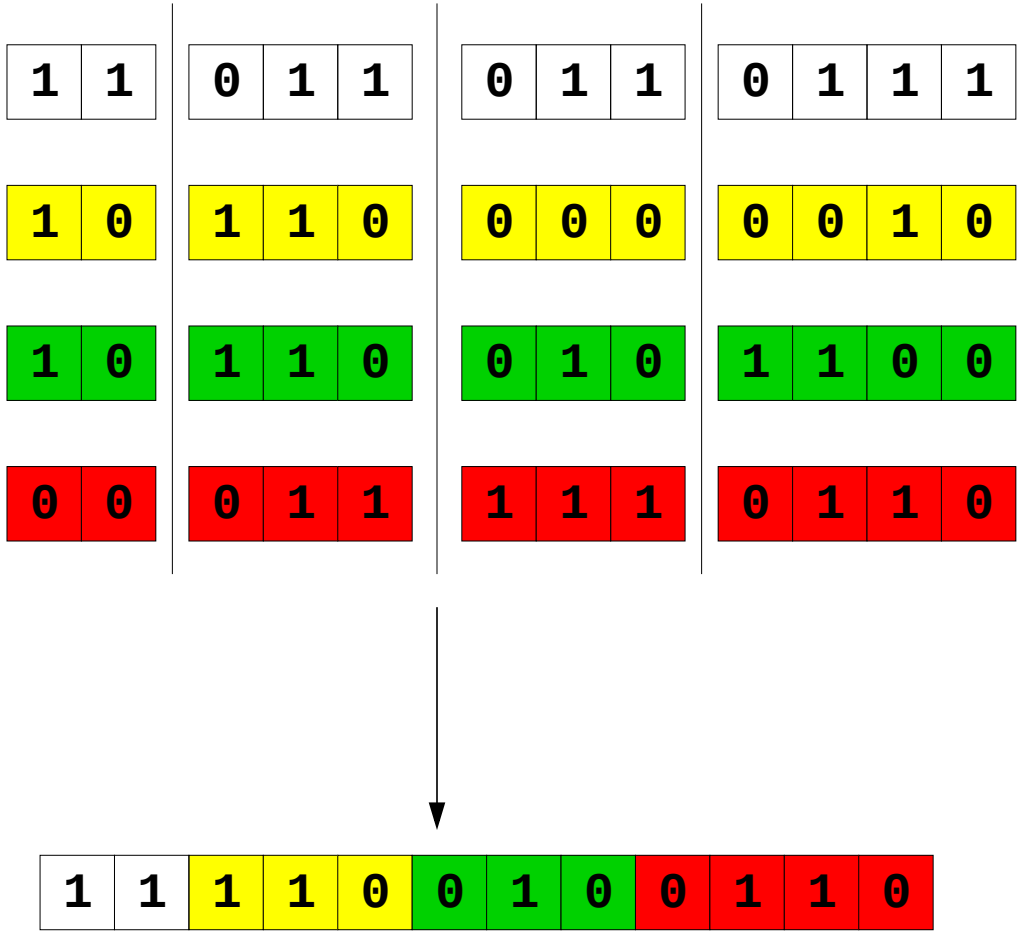
Recombination:

- *Multi-point crossover* (recombination of more than 2 “pieces” of chromosomes)
- *Multi-parent crossover* (the genetic material of a new individual is taken from more than 2 parents)
- *Uniform crossover* (children created by randomly shuffling the parent variables at each site)

Multi-point crossover



Multi-parent crossover



Toward *less simple* GA

Selection:

- Different probability distribution (e.g., probability distribution based on the *ranking* of individuals)
- *Tournament Selection* (iteratively pick two or more individuals and put in the *mating pool* the fittest)

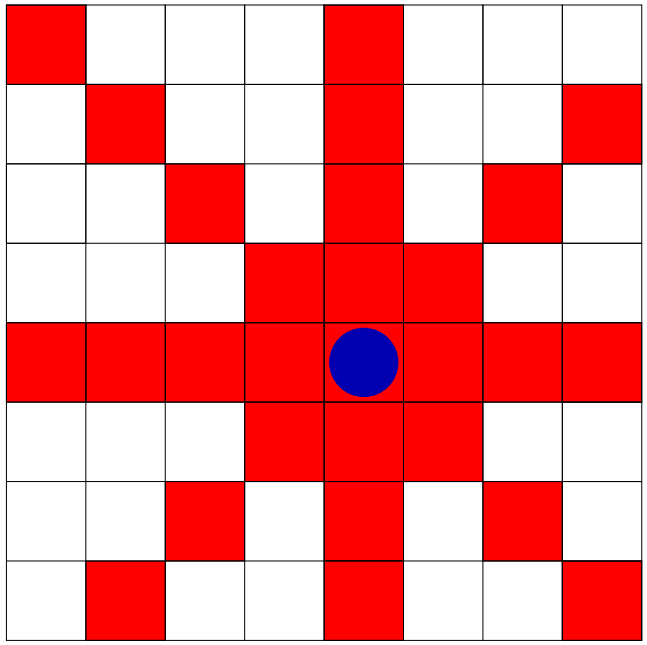
Ex: real valued variables

- Solution: $x \in [a, b],\ a, b \in \mathbb{R}$
- Mutation: random perturbation $x \rightarrow x \pm \delta$, accepted if $x \pm \delta \in [a, b]$
- Crossover: linear combination $z = \lambda_1 x + \lambda_2 y$, with λ_1, λ_2 such that $a \leq z \leq b$.

Example: permutations

- Solution: $x = (x_1, x_2, \ldots, x_n)$ is a permutation of $(1, 2, \ldots, n)$.
- Mutation: random exchange of two elements in the n-ple.
- Crossover: like 2-point crossover, but avoiding value repetition.

Eight Queens

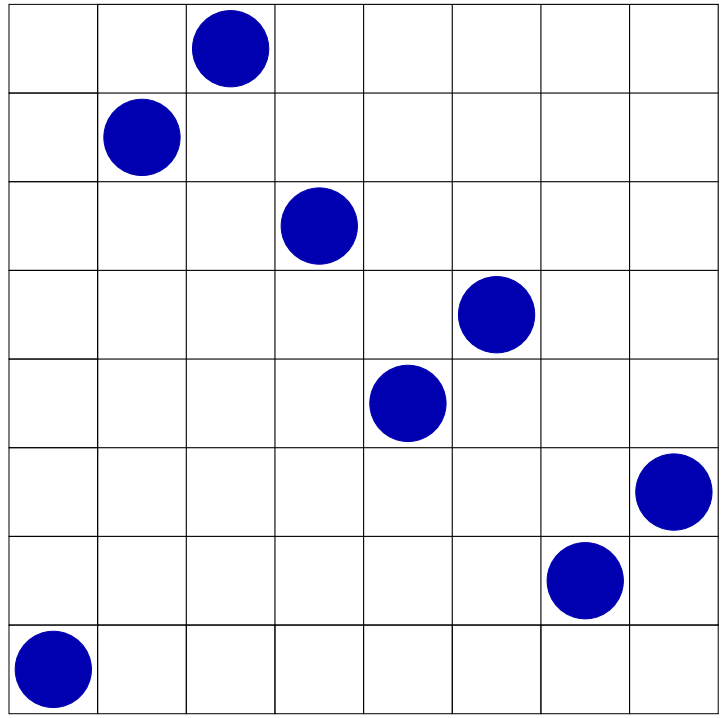


Place 8 queens on a 8×8 chessboard in such a way that the queens cannot attack each other.

Eight Queens

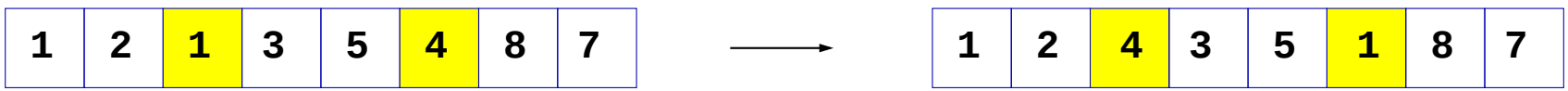
Genotype: a permutation of the numbers 1 through 8

3	2	4	6	5	8	7	1



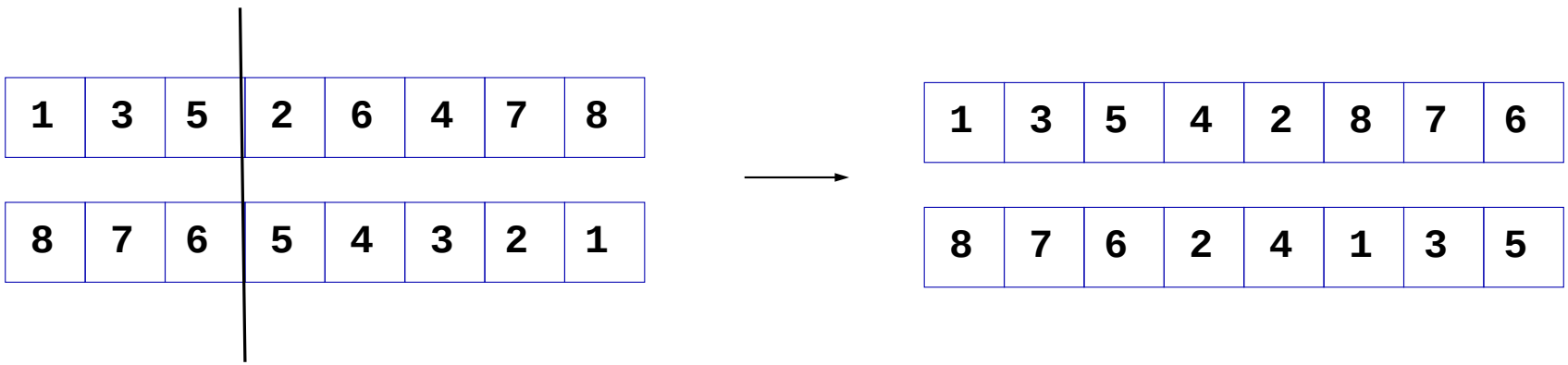
Eight Queens

Mutation: swap two numbers



Eight Queens

Crossover: combine two parents



Eight Queens

Fitness: penalty of a queen is the number of queens it can check.

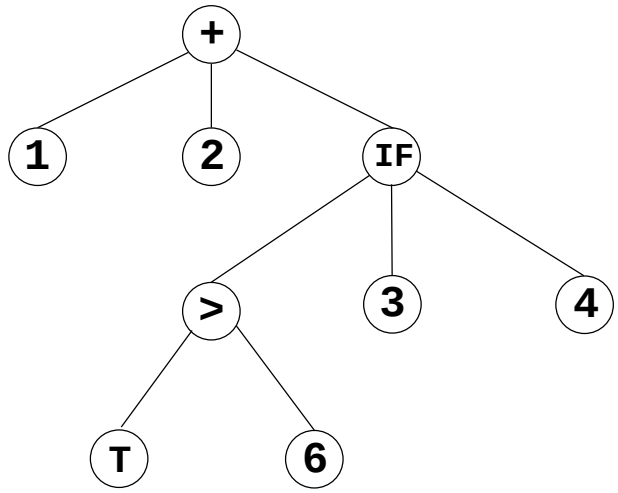
The fitness of the configuration is the sum of the single penalties.

Genetic Programming

- Can be seen as a ‘variant’ of GA: individuals are **programs**.
- Used to build programs that solve the problem at hand ($\Rightarrow$ specialized programs).
- Extended to *automatic design* in general (e.g., controllers and electronic circuits).
- Fitness is given by evaluating the performance of the program (based upon some defined criterion).

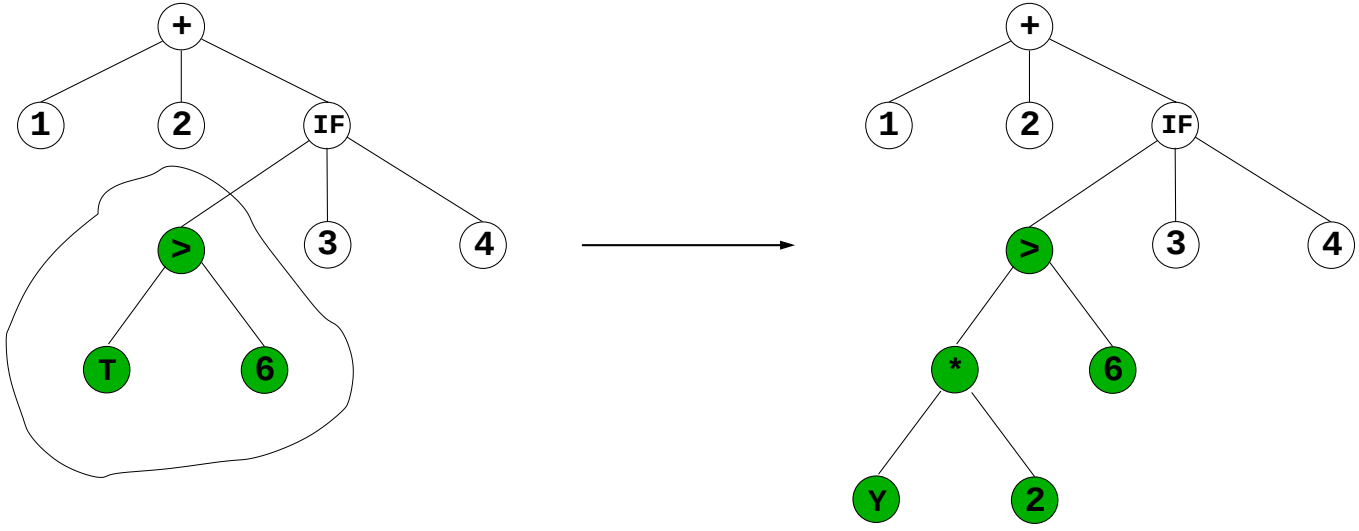
Genetic Programming

In most of the cases, individuals are represented as trees, which encode programs.



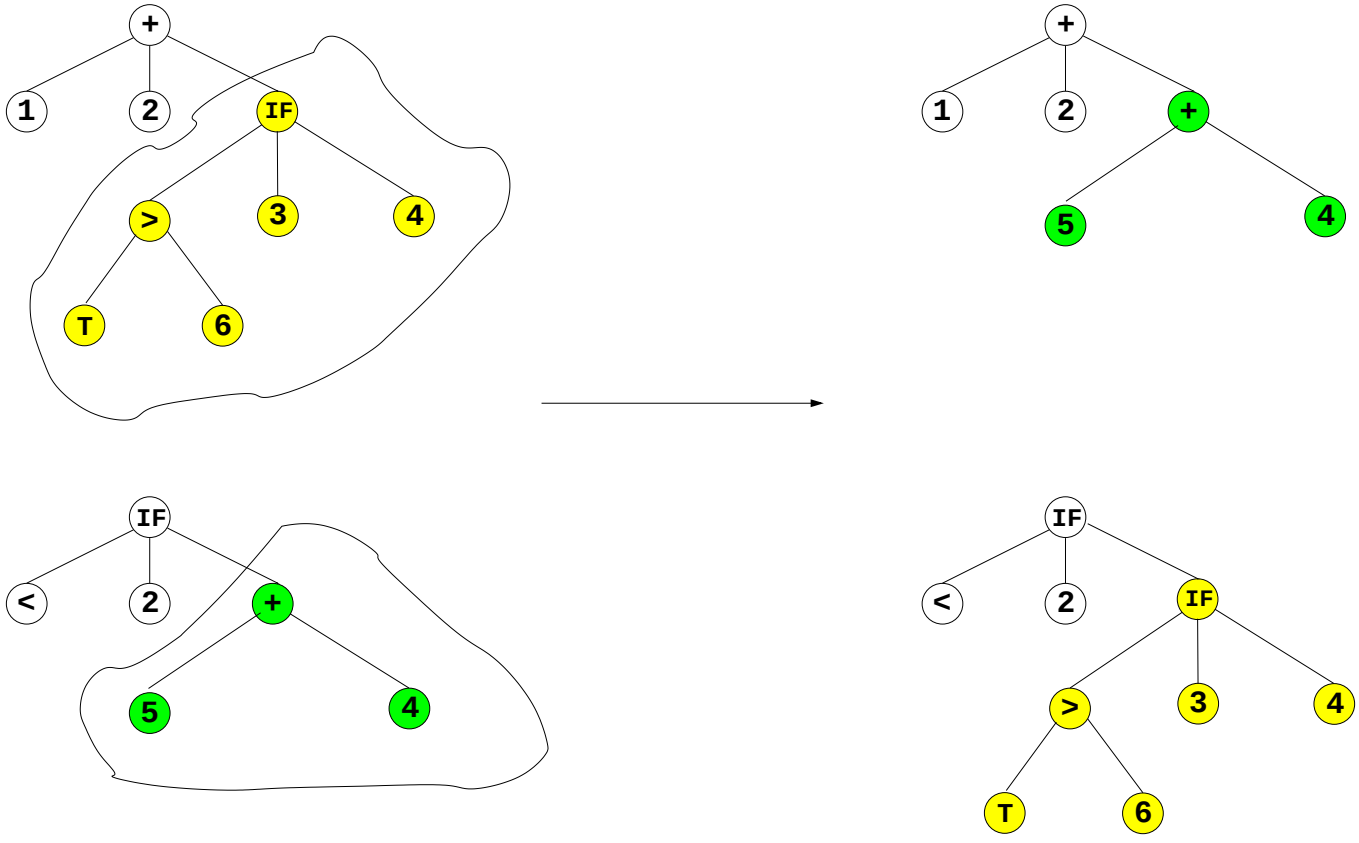
Operators

Mutation: Random selection of a subtree which is substituted by a *well formed* random generated subtree.



Operators

Crossover: Swap two randomly picked subtrees.



The realm of GP

- *Black art* problems.
  E.g.,automated synthesis of analog electrical circuits, controllers, antennas, and other areas of design.
- *Programming the unprogrammable*, involving the automatic creation of computer programs for unconventional computing devices.
  E.g.,cellular automata, parallel systems, multi-agent systems, etc.

Coevolution

Species evolve in the same environment

$\longrightarrow$ ***dynamic* environment**

Two kinds:

- Competitive
- Cooperative

Competitive Coevolution

- Species evolve trying to face each other
  - E.g., prey/predator, herbivore/plants.

Applications: ALU design for Cray computer, (pseudo-)random number generator.

Cooperative Coevolution

▷ Species evolve complementary capabilities to survive in their environment

- E.g., host/parasite.

Applications: ‘niche’ genetic algorithms for *multi-criteria* optimization.

Examples

The Prisoner's Dilemma

Axelrod and The Prisoner's Dilemma

- Game strategies evolved through genetic algorithms.
- Dynamic environment (a player plays against other different players).
- Best strategy evolved by GA is the best human strategy.
- Analysis of the arising of cooperation.

The Prisoner's Dilemma

- The two players in the game can choose between two moves, either *cooperate* or *defect*.
- Each player gains when both cooperate, but if only one of them cooperates, the other one, who defects, will gain more.
- If both defect, both lose (or gain very little) but not as much as the ”cheated” cooperator whose cooperation is not returned.

The payoff matrix

	Cooperate	Defect
Cooperate	R,R	S,T
Defect	T,S	P,P

T: Temptation to defect

R: Reward for mutual cooperation

P: Punishment for mutual defection

S: Sucker's payoff

HP: $T > R > P > S$; $2R > T + S$

Problem encoding

Suppose that the memory of each player is one previous move. E.g., player A cooperated and player B defected becomes: `CD`.

The strategy is defined with a move for each possible past move. E.g.:

```
If CC then C
If CD then D
If DC then C
If DD then D
```

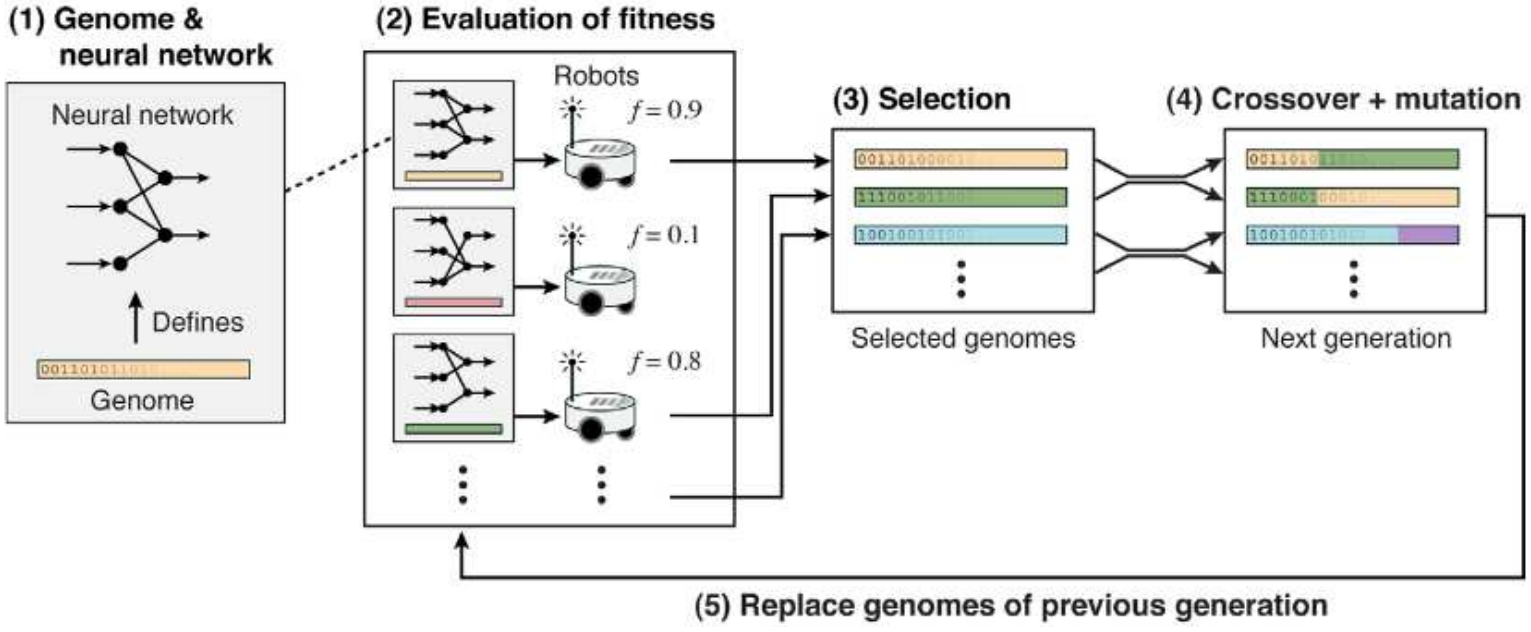
$\longrightarrow$ the string is `CDCD`

Tit for tat strategy (winner).

Evolutionary robotics

- Robots are controlled by means of neural networks.
- The neural network is designed by means of an EC technique.
- The fitness is computed by simulating the robot.
- The best resulting robot’c controller is tested in a real setting.

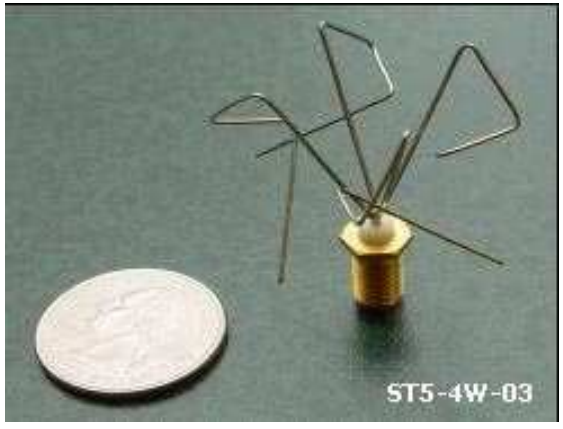
Evolutionary robotics



(taken from D. Floreano and L. Keller, *Evolution of Adaptive Behaviour in Robots by Means of Darwinian Selection*, PLOS Biology, Jan. 2010, Vol. 8, Issue 1)

NASA antenna design

- Space Technology 5 Project.
- Antennas are defined through a LOGO-like programming language.
- Antenna construction programs are evolved by means of an EC technique.



EC and Artificial Life

Tierra

- Artificial evolution of computer programs (T. Ray, early ’90s)
- Environment: virtual computer
- Individuals: self-replicating assembler programs
- Resources: CPU time and memory

Tierra

Results of evolution: several kinds of nontrivial behaviors and dynamics

- parasites
- immunity to parasites
- circumvention of immunity to parasites
- social individuals
- . . . and others

Suggested references

- M.Mitchell. *Genetic Algorithms*. MIT Press, 1999.
- D.E.Golberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, 1989.
- R. Poli, W.B. Langdon, N.F. McPhee, J. Koza. *A Field Guide to Genetic Programming*, http://www.gp-field-guide.org.uk/ (free download –Creative Commons).
- S. Nolfi and D. Floreano, *Evolutionary robotics*. The MIT Press, 2000.