An introduction to evolutionary computation

Andrea Roli
andrea.roli@unibo.it

Dept. of Computer Science and Engineering (DISI)
Campus of Cesena
Alma Mater Studiorum Università di Bologna

Outline

Inspiring principle

Evolutionary Computation is inspired by **natural evolution**

Three observations:

- *Adaptation*: organisms are suited to their habitats
- *Inheritance*: offspring resemble their parents
- *Natural selection*: new, adapted types of organisms emerge and those that fail to change adequately are subject to extinction

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
- Estimation of distribution algorithms
- ...

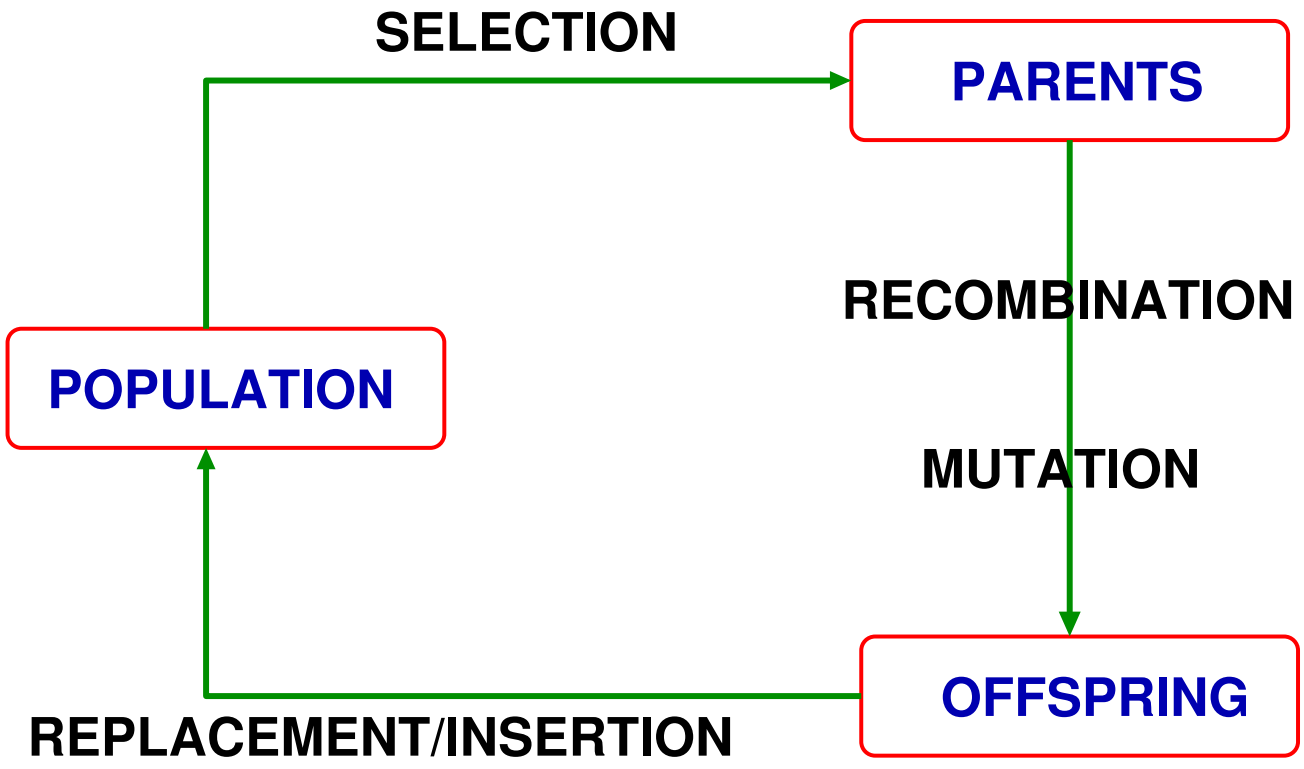
Main applications

- System design
- Neural network training
- Signal processing
- Optimization (discrete and continuous)
- Robotics
- 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

Mutation: introduce variability in the genotypes.

Recombination: combines the genetic material of the parents.

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

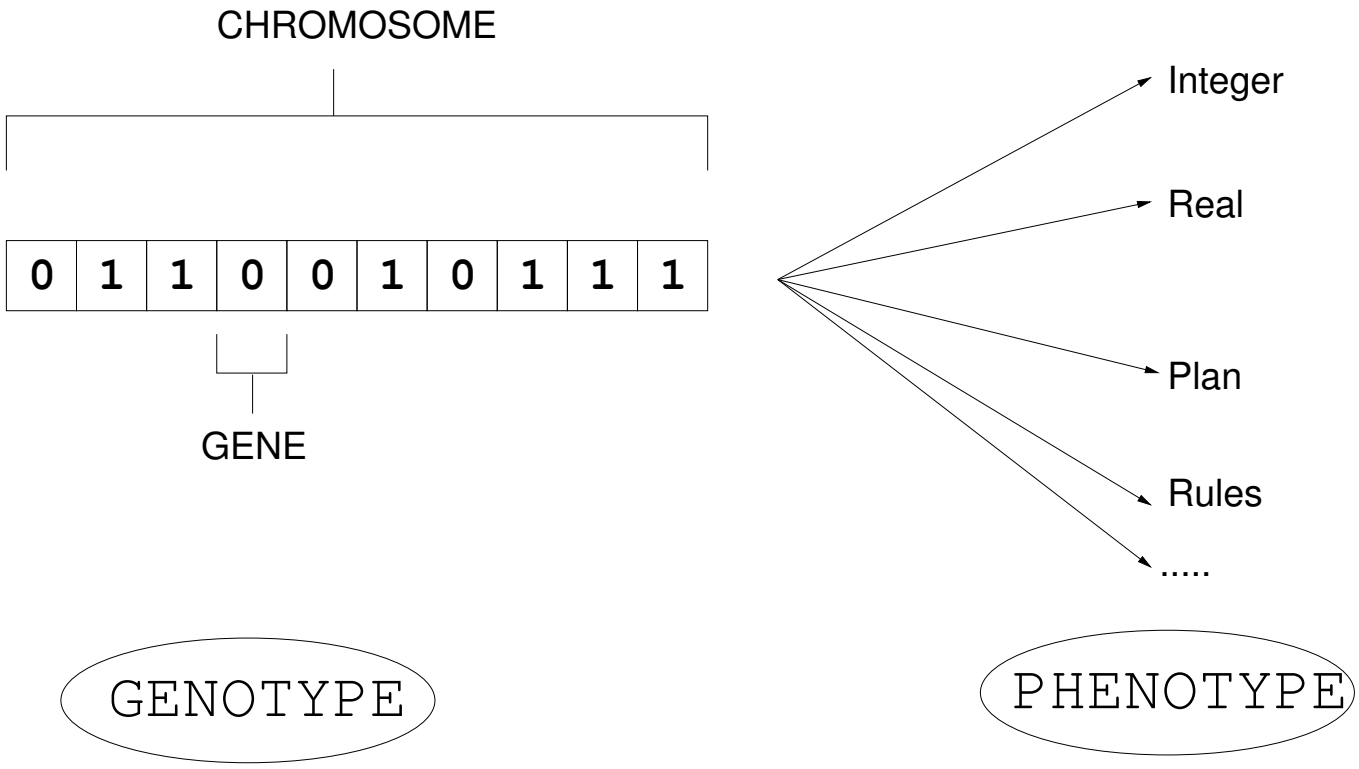
- Directly derived from the natural metaphor
- Very simple model
- Programming oriented

A bit of terminology

- A **population** is the set of individuals (possible 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:

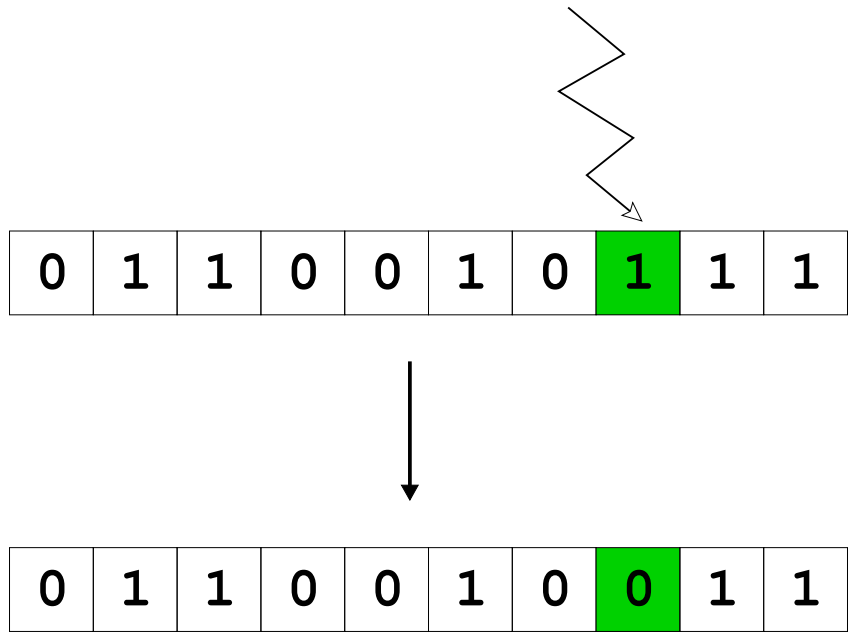
- binary coding $\rightarrow$ string of 7 bit;
- 4 bits per digit $\rightarrow$ string of 12 bit.

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

- binary coding $\rightarrow$ string

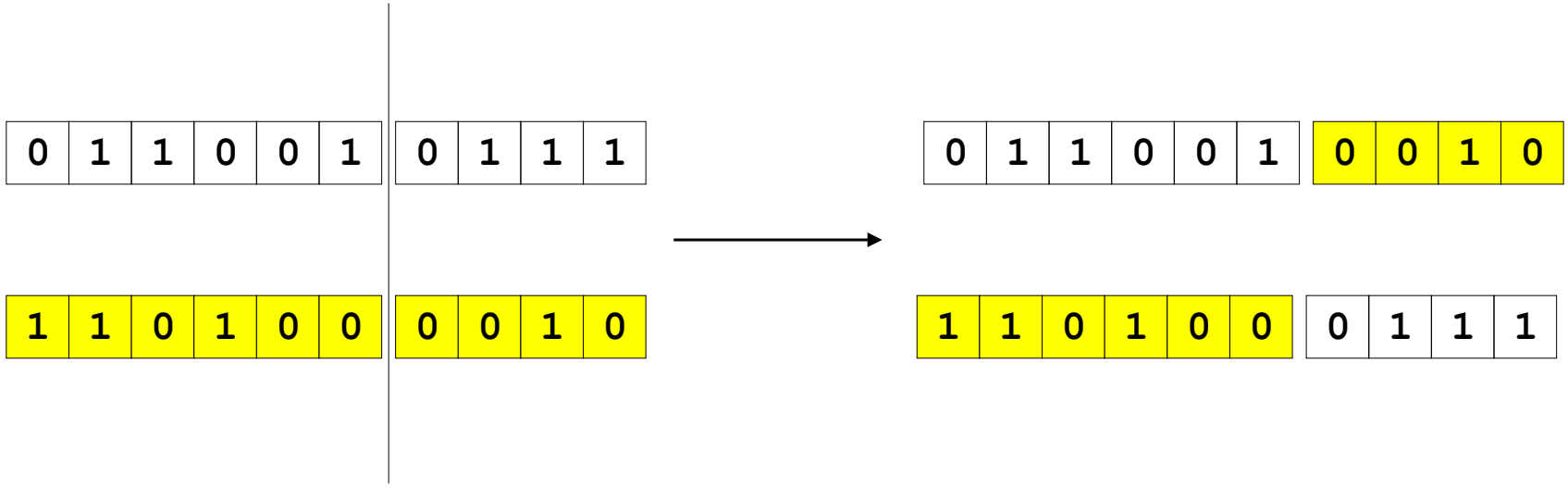
SGA genetic operators (1)

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



SGA genetic operators (2)

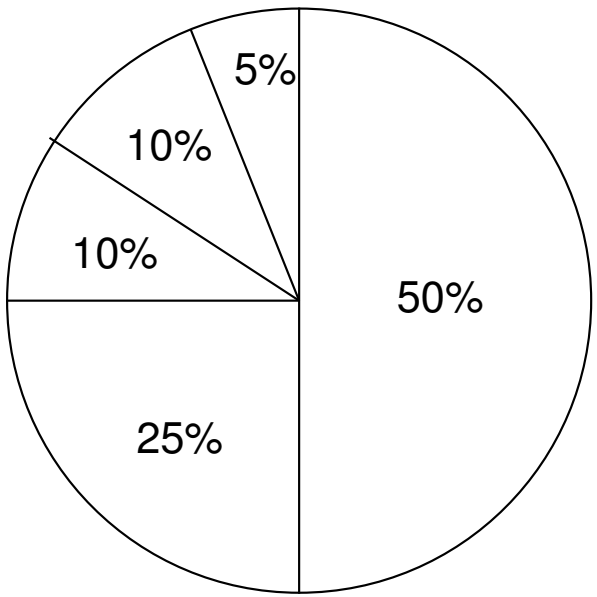
Recombination or **Crossover**: cross-combination of two chromosomes (loosely resembling biological crossover)



SGA genetic operators (3)

$\rightarrow$ **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: good solutions might not be 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
    Copy their chromosomes {Note: the parents are not deleted from the population}
    Apply crossover to produce two new individuals
    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)

Example by hand

- Problem: find the chromosome with minimal distance from $11\ldots1$
- The fitness is given by the number of bits with value 1 (one among many possible ways for defining it)

Example by hand

We tackle the problem with a SGA with the following setting:

- length of chromosome $n = 8$
- populations size $\mathrm{popsize} = 4$
- One-point crossover (always applied)
- Mutation: each bit has a probability p_m of being flipped. We set $p_m = 1/n$ such that we have on average one flip per chromosome (there are of course many other ways to define it)
- Selection: proportional
- New generation obtained with generational replacement (according to the SGA scheme, the new population is the offspring)

Example by hand

Initial random population:

```
00010110
00011100
11001011
00001001
```

Evaluation (with fitness and corresponding probabilities):

```
00010110 : 3 : 3/13
00011100 : 3 : 3/13
11001011 : 5 : 5/13
00001001 : 2 : 2/13
```

Example by hand

Generate two new chromosomes:

1) Select two chromosomes with a proportional rule:

```
00010110 : 3 : 3/13
11001011 : 5 : 5/13
```

2)Apply crossover (random crossover point):

```
000101.10    000101.11
          ->
110010.11    110010.10
```

3)Apply mutation

```
00010111 -> 01010111
11001010 -> 01000010
```

Example by hand

Generate the other two chomosomes:

1) Select two chromosomes with a proportional rule:

```
11001011 : 5 : 5/13
11001011 : 5 : 5/13
```

2)Apply crossover (random crossover point):

```
110.01011    110.01011
          ->
110.01011    110.01011
```

3)Apply mutation

```
11001011 -> 11001010
11001011 -> 11001111
```

Example by hand

New population and its evaluation:

```
01010111 : 5 : 5/17
01000010 : 2 : 2/17
11001010 : 4 : 4/17
11001111 : 6 : 6/17
```

...and we go on with generations until a termination condition is verified.

Observations

- Genetic operators are blind with respect to the goal
- Their role is to recombine and variate the genetic material so as to *explore* genetic combinations
- Favourable genetic variations are likely to be preserved by selection
- Nevertheless, because of generational replacement, it is possible that good individuals are not kept in the new population. Therefore, a kind of *elitism* is always used

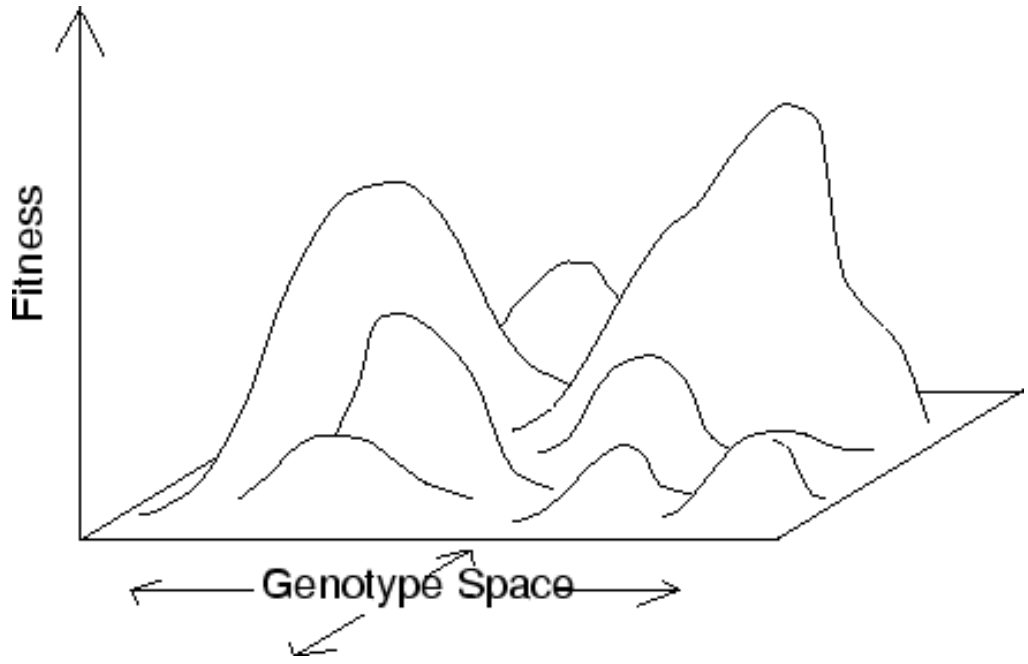
Why does it work?

Intuition:

- Crossover combines good parts from good solutions (but sometimes 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 exist.

The search process

Intuition:

- the GA performs an iterative sampling over the search space
- sampling is based on a probabilistic model
- the parameters of the model are tuned as a function of the population fitness, so as to intensify the sampling around promising regions

Process similar to importance sampling and *model based search*.

Schemata and building blocks

Holland provides a formal explanation of why the SGA 'works' in terms of:

- **Schemata**
- **Building blocks**

Schemata

- A *schema* is a kind of mask: $001*1**0$
- The symbol $*$ represents a wildcard: both 0 and 1 fits
- So, $1*0$ represents 100 and 110

Building blocks

- A *building block* is a pattern of contiguous bits
- HP: good solutions are composed of good building blocks
- The crossover puts together short building blocks and destroys large ones

Implicit parallelism

- Every individual corresponds to a set of schemata
- The number of the best schemata increases exponentially
- The solution space is searched through schemata (hence implicit parallelism)

Impact of building blocks

A SGA works well if:

1. Short good building blocks (correlate genes are adjacent)
2. Loose interaction among genes (low *epistasis*)

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.

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 mixing parent's gene: each parent k is associated a probability g_k which is then used to pick the value of gene i in the child)

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)

New population:

- *Steady state* scheme: the new population is composed of the best individuals from the current population and the offspring
- Elitism: like replacement, but we keep the λ best individuals from the current population (in case of fixed size populations, the worst λ individuals of the offspring are removed)

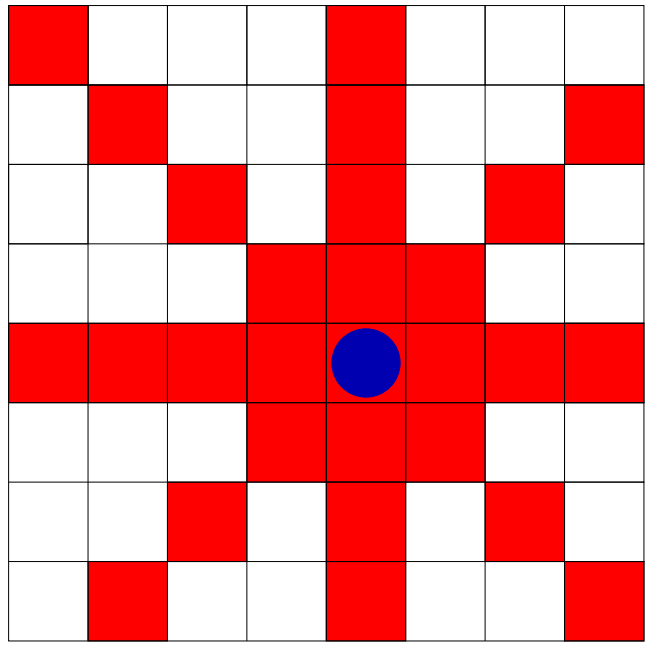
Ex: real valued variables

- Solution: $x \in [a, b],\ a, b \in \mathbb{R}$
- Mutation: random perturbation $x \to x \pm \delta$, 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.

n-Queens

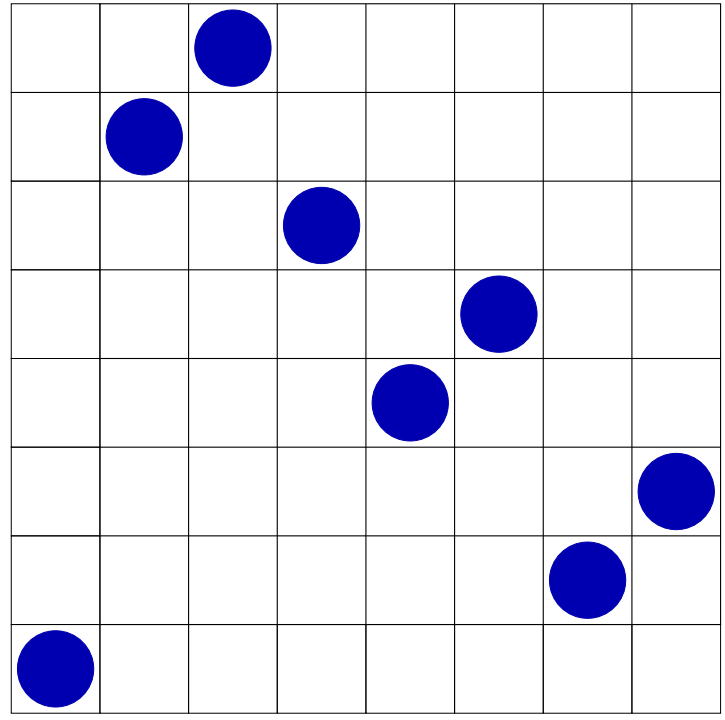


Place n queens on a $n \times n$ chessboard in such a way that the queens cannot attack each other.

n-Queens

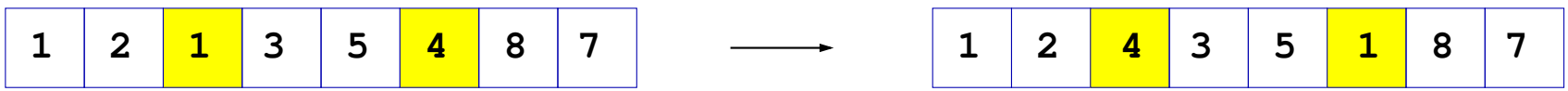
Genotype: a permutation of the numbers 1 through n

3	2	4	6	5	8	7	1



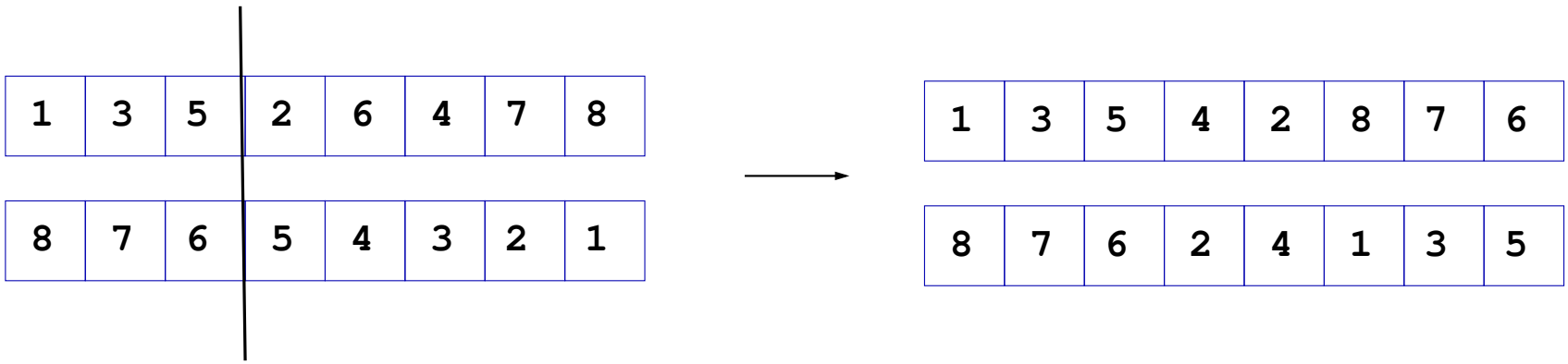
n-Queens

Mutation: swap two numbers



n-Queens

Crossover: combine two parents



n-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.

GA design guidelines

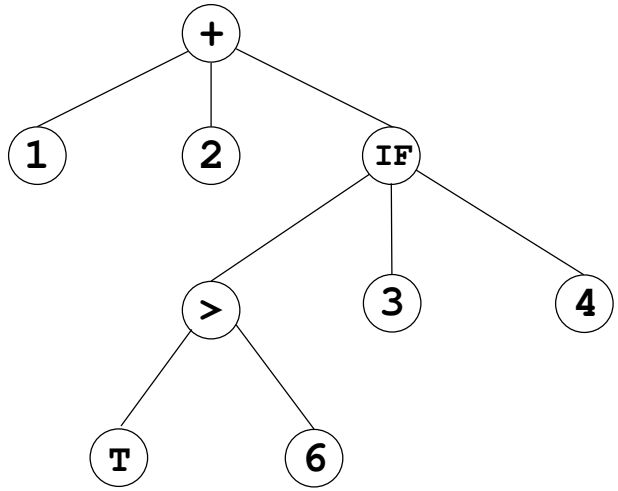
- Always use mutation and make this operator tunable so as to introduce controlled randomness.
- Use recombination operators only if they are able to combine good solution parts. If not, do not use them.
- Prefer tournament selection first, the try other mechanism.
- Carefully design the fitness function so as to attain a fitness-distance correlation (i.e., nearby solutions have close fitness values).

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).

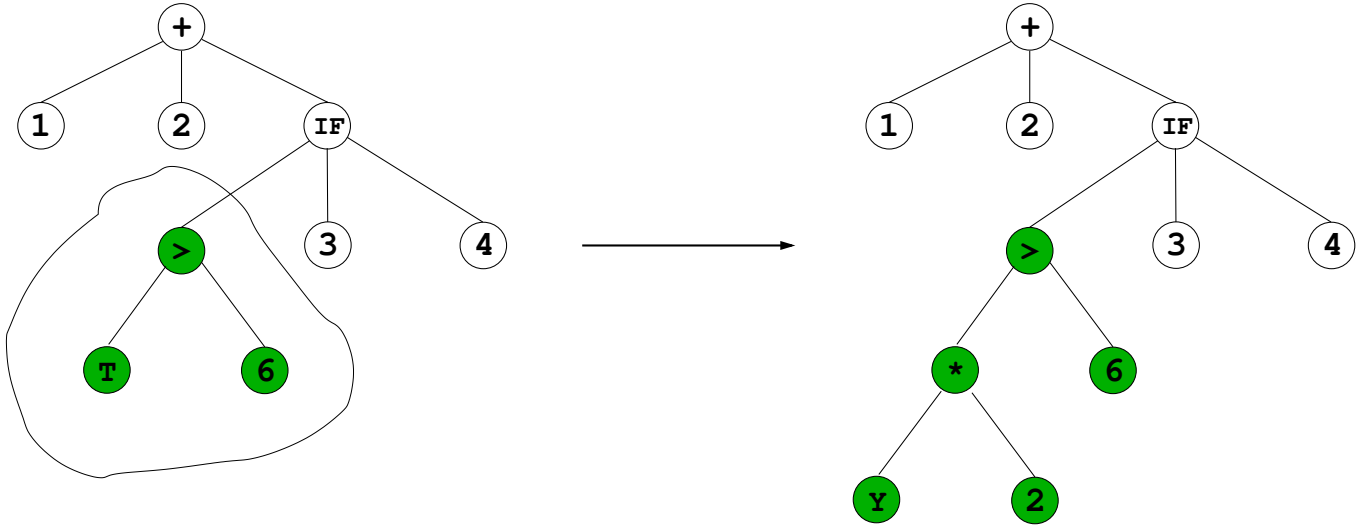
Genetic Programming

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



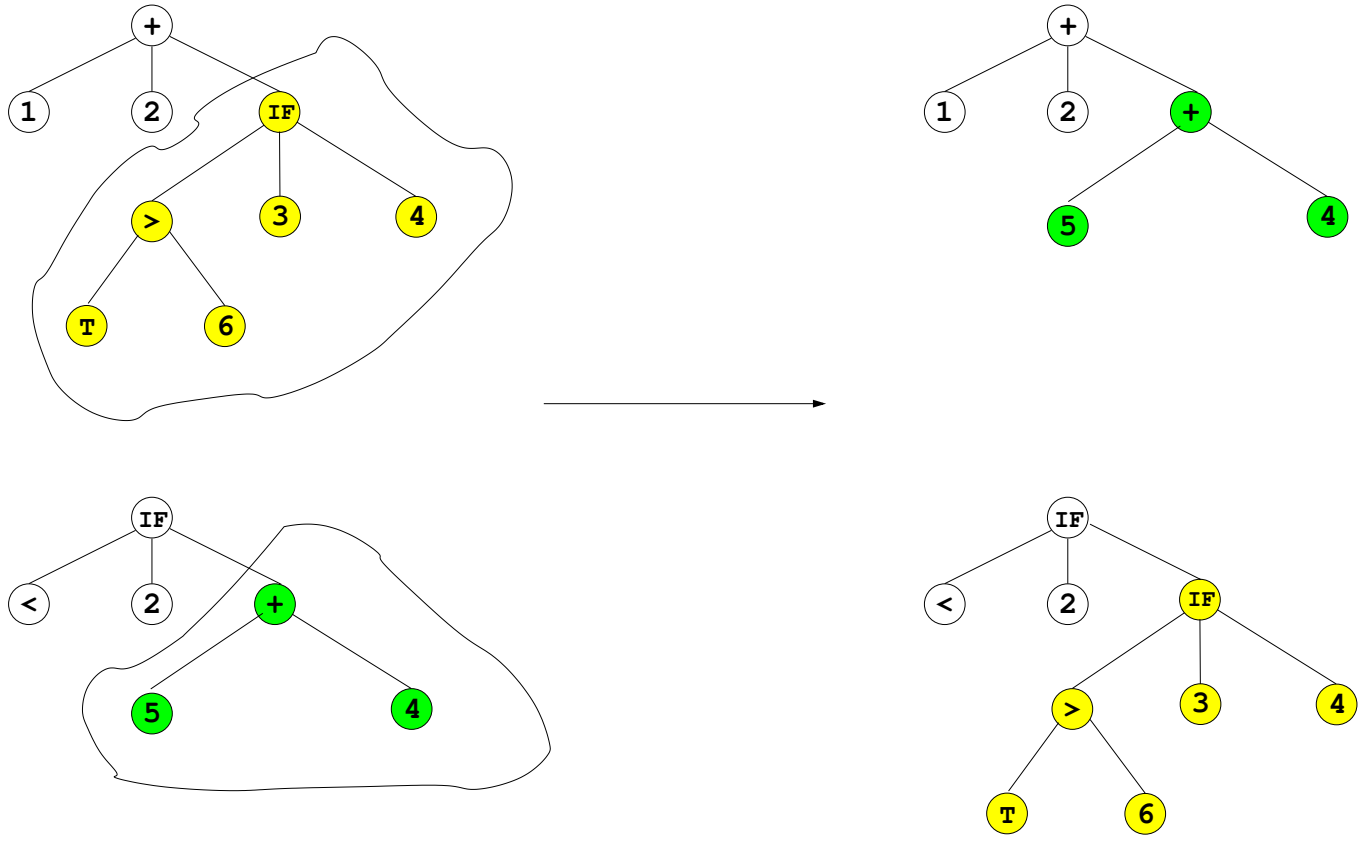
Operators

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

$\rightarrow$ **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.

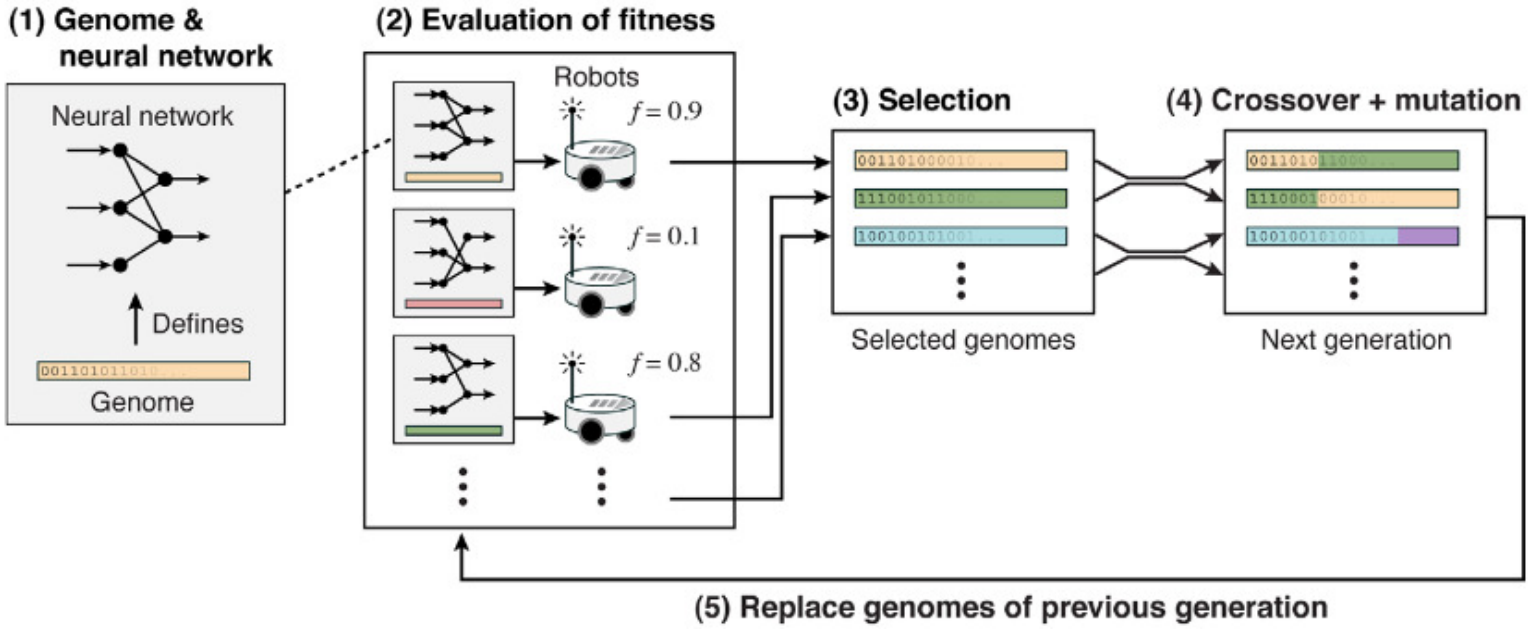
Applications: 'niche' genetic algorithms for *multi-criteria* optimization.

Examples

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 controller is tested in a real setting.

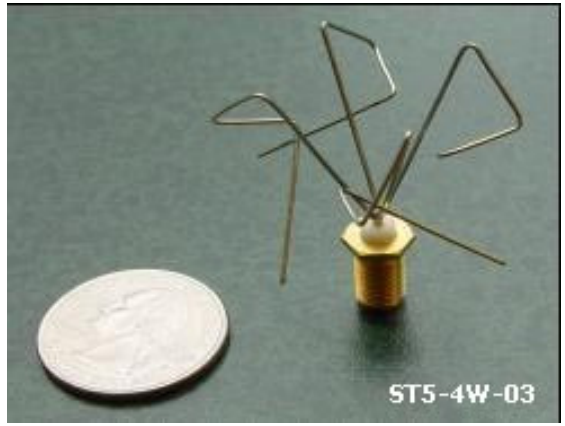
Evolutionary robotics



(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 for repairing and testing

- EC can be applied to generate sets of tests for system testing and verification. E.g.: hard test instances for optimisation algorithms or tests for circuits.
- EC can also be used to repair software, as in GenProg (`http://dijkstra.cs.virginia.edu/genprog/`)

Suggested bibliography (1/2)

- M.Mitchell. *Genetic Algorithms*. MIT Press, 1999.
- Z.Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs*, Springer, 1992.
- D.E.Golberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, 1989.
- W.B.Langdon, R.Poli. *Foundations of Genetic Programming*. Springer, 2001.

Suggested bibliography (2/2)

- 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.
- A.E. Eiben & J.E. Smith, *Introduction to evolutionary computing*. Springer, 2nd. edition, 2015.