# Evolutionary computation


Andrea Roli

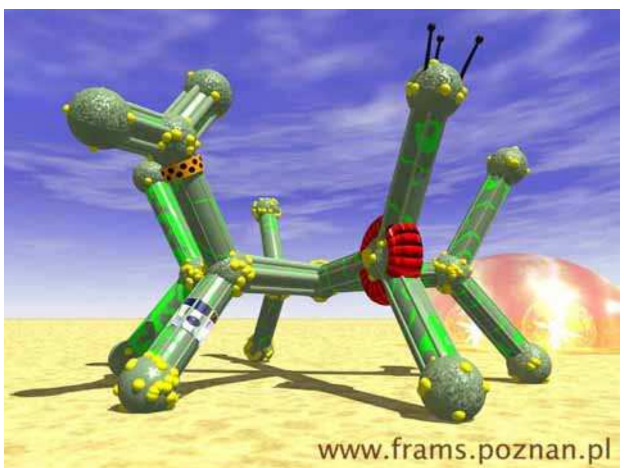
andrea.roli@unibo.it

DEIS

*Alma Mater Studiorum* Università di Bologna




# Evolutionary Computation






# Evolutionary Computation

Inspiring principle: theory of **natural selection**

“Species face the problem of searching for beneficial adaptations to the environment. The *knowledge* that each species has gained is embodied in the makeup of the chromosomes of its members.” (Davis, *Genetic Algorithms and Simulated Annealing*, 1987)

Example: rabbits. . .



# Evolutionary Computation

Evolutionary Computation (EC) encompasses:

- Genetic Algorithms
- Genetic Programming
- Evolution Strategies
- Estimation of Distribution Algorithms

# Objectives

- Problem solving
- Optimization
- Adaptive systems design
- Simulation



# Some applications

- ▷ System design (e.g., airplanes, electronic circuits, mechanical elements)
- ▷ Neural network training (e.g., robotics)
- ▷ Signal processing (e.g., artificial vision)
- ▷ Optimization (discrete and continuous)



# More applications

- ▷ Time series analysis and forecasting (e.g., financial forecasting)
- ▷ Artificial Life (e.g., cellular automata, analysis of complex adaptive systems)
- ▷ Games (e.g., Prisoner's Dilemma)

Challenge: find a problem where EC has NOT been applied!



# Genetic Algorithms

## The Metaphor

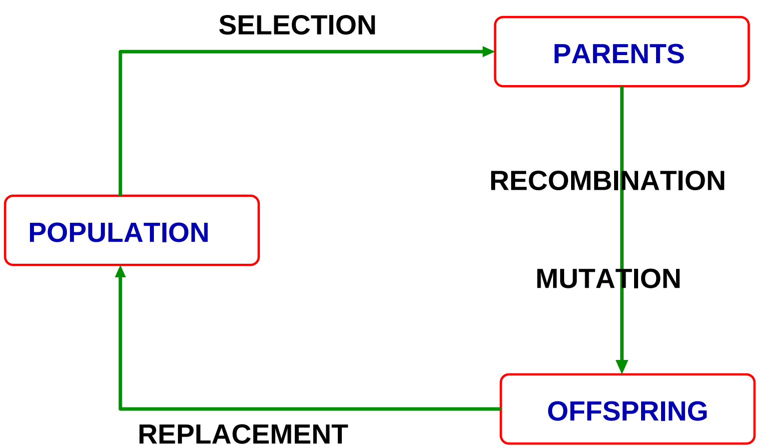
| NATURAL EVOLUTION | | ARTIFICIAL SYSTEMS |
|---|---|---|
| Individual | $\leftrightarrow$ | A possible solution |
| Fitness | $\leftrightarrow$ | Quality |
| Environment | $\leftrightarrow$ | Problem |

# A bit of terminology

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



# The Evolutionary Cycle






# Genetic operators

- Mutation
- Recombination
- Selection
- Replacement/insertion



# Genetic operators

▷ EC algorithms define a basic computational procedure which uses the genetic operators.

▷ The definition of the genetic operators specifies the actual algorithm.

▷ The definition of the genetic operators 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



# Simple Genetic Algorithm

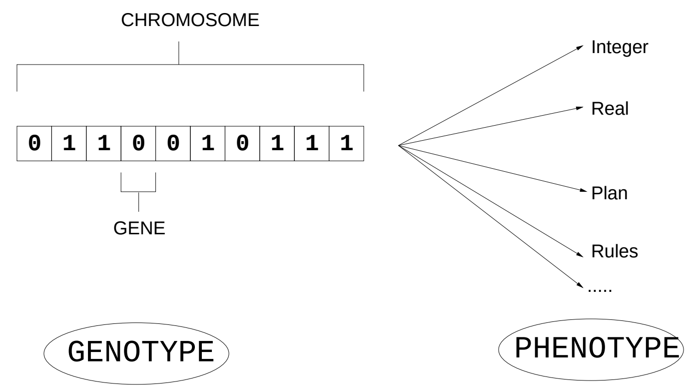
- Derived from the natural metaphor
- Very simple model
- ‘Programming oriented’

You can take it as a first step toward evolutionary algorithms in general



# Simple Genetic Algorithm

Solutions are coded as **bit strings**





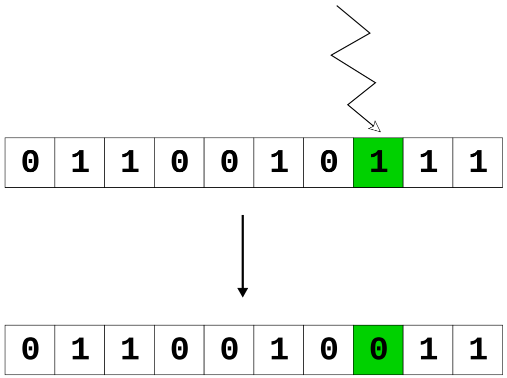

# Example

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

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

# Genetic operators (1)

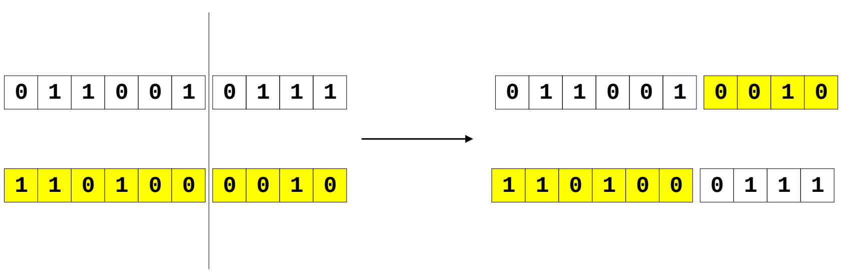
**Mutation**: each gene has probability $p_M$ of being modified ('flipped')






# Genetic operators (2)

**Crossover**: cross-combination of two chromosomes (loosely resembling human crossover)






# Genetic operators (3)

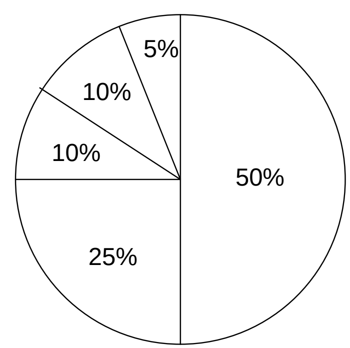
**Selection** acts in the choice of parents and produces the *mating pool*.

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



# Genetic operators (3)

***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 (extremely) expensive, easier theoretical analysis.
- Disadvantage: we could loose good solutions



# 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

The basic question is: **when to stop?**

- Execution time limit reached
- We are satisfied with the solution(s) obtained
- Stagnation (limit: the population converged to the same individual)



# Simple Genetic Algorithm

**Initialize Population**{$N_{pop}$ individuals $X_1, \ldots, X_{N_{pop}}$}
**for** $i = 1$ to $N_{pop}$ **do**
    $X_i \leftarrow$ InitialSolution() {e.g., random}
**end for**

**Evaluate Population**{Individual $X_i$ has fitness $F_i$}
**for** $i = 1$ to $N_{pop}$ **do**
    $F_i \leftarrow$ Eval($X_i$)
**end for**

# Simple Genetic Algorithm

**Select parents:** $G_1, G_2$ {Roulette wheel selection}

$lung \leftarrow 0$

**for** $i = 1$ to $N_{pop}$ **do** {all fitness values are summed up}

  $lung \leftarrow lung + F_i$

**end for**

**for** $m = 1$ to $2$ **do**

  $r \leftarrow$ Random(0,$lung$); $sum \leftarrow 0$; $i \leftarrow 1$

  **while** $i < N_{pop}$ AND $sum < r$ **do**

    $sum \leftarrow sum + F_i$; $i \leftarrow i + 1$

  **end while**

  $G_m \leftarrow X_i$

**end for**



# Simple Genetic Algorithm

**Apply crossover**: from $G_1, G_2$ we get $G'_1, G'_2$

$r \leftarrow$ Random(1,$l_{chromosome}$){crossover point}

**for** $i = 1$ to $r - 1$ **do**

  $G'_1[i] \leftarrow G_1[i]$

  $G'_2[i] \leftarrow G_2[i]$

**end for**

**for** $i = r$ to $l_{chromosome}$ **do**

  $G'_1[i] \leftarrow G_2[i]$

  $G'_2[i] \leftarrow G_1[i]$

**end for**



# Simple Genetic Algorithm

**Apply mutation** to individual $X$

**for** $i = 1$ to $l_{chromosome}$ **do**

  $r \leftarrow$ Random(0,1)

  **if** $r \leq p_M$ **then**

    Complement $X[i]$

  **end if**

**end for**



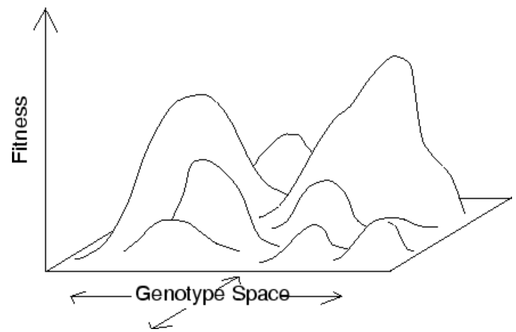
# SGA: Example

Maximization of a real function

Taken from: `http://www.evonet.polytechnique.fr/CIRCUS2/`

# Fitness Landscape

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






# Fitness Landscape

Caution!

- Different landscapes for different operators
- In many cases fitness landscapes are *dynamic*
- Landscape ‘intuition’ might be misleading
- Use of term *local optimum* used and abused everywhere



# Why does it work?

Intuition:

- Crossover combines good parts from good solutions (but it might also destroy. . . sometimes)
- Mutation introduces diversity
- Selection drives the population toward high fitness



# SGA: pros and cons

Pros:

- Extremely simple
- General purpose
- Theoretical models

Cons:

- Coding
- Too simple genetic operators

# A recipe

The ingredients to prepare a GA:

- Solution coding (e.g., bit strings, programs, arrays of real variables, etc.)
- Define a way of evaluating solutions (e.g., objective function value, result of a program, behavior of a system, etc.)
- Define recombination operators (*crossover*, *mutation*)
- Define the selection and replacement/insertion mechanisms



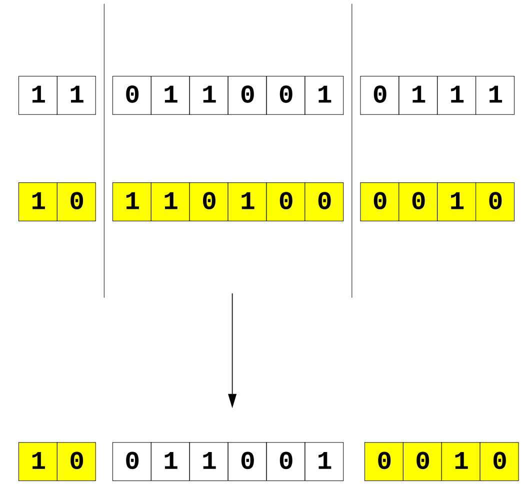
# Toward *less simple* GA

Recombination:

- *Multi-point crossover* (recombination of more than 2 “pieces” of chromosomes)
- *Multi-parent crossover* (an individual is generated by more than 2 parents)
- *Uniform crossover* (children created by randomly shuffling the parent variables at each site)

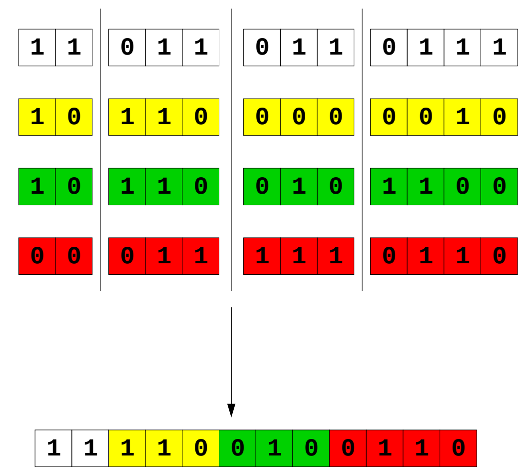


# Multi-point crossover






# Multi-parent crossover



# Toward *less simple* GA

Mutation:

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

Interesting topic: Evolution & Learning,

`www.cogs.susx.ac.uk/users/ezequiel/alife-page/evolearn.html`



# 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

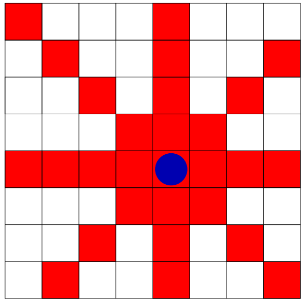
- Solution: $x \in [a, b], \ a, b \in \mathbb{R}$
- Mutation: random perturbation $x \to x \pm \delta$, accepted if $x \pm \delta \in [a, b]$
- Crossover: linear combination $z = \lambda_1 x + \lambda_2 y$, with $\lambda_1, \lambda_2$ such that $a \leq z \leq b$.



# 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 (see next example).

# Eight Queens



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

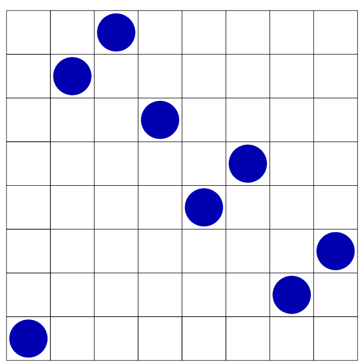


# Eight Queens

Genotype: a permutation of the numbers 1 through 8

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 3 | 2 | 4 | 6 | 5 | 8 | 7 | 1 |

→





# Eight Queens

Mutation: exchanging two numbers

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | 2 | 1 | 3 | 5 | 4 | 8 | 7 |

→

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | 2 | 4 | 3 | 5 | 1 | 8 | 7 |



# Eight Queens

Crossover: combining two parents

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | 3 | 5 | 2 | 6 | 4 | 7 | 8 |
| 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 |

→

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | 3 | 5 | 4 | 2 | 8 | 7 | 6 |
| 8 | 7 | 6 | 2 | 4 | 1 | 3 | 5 |

# 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.



# Example

Traveling Salesman Problem

Taken from:
`http://ouray.cudenver.edu/∼da0todd/neural/third_homework/dave/test/TSP_Genetic_Algorithm.htm`



# Mondriaan Art

Mondriaan Art

Taken from: `http://www.evonet.polytechnique.fr/CIRCUS2/`

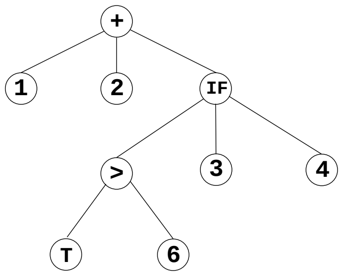


# Genetic Programming

- 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)

# Genetic Programming

Individuals are *trees* which encode programs.

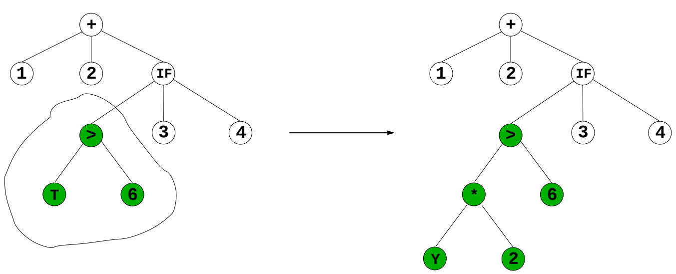



Fitness given by the evaluation of the program "behavior" (based upon some defined criteria)



# Operators

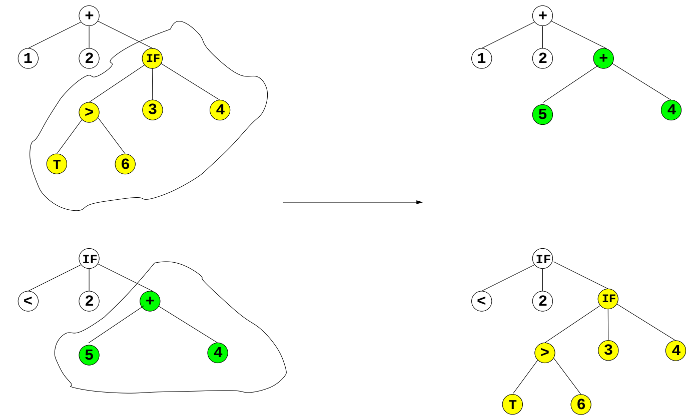
**Mutation**: Random selection of a subtree which is substituted by a *well formed* random generated subtree






# Operators

**Crossover**: Exchange two randomly picked subtrees.






# Operators

**Selection and replacement**

Fitness is evaluated depending on the application.

- For *assembler worms* the fitness can be the memory they occupied.
- For controllers, the fitness can be the percentage of correct actions

# 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

→ ***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.

# Some references

- 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.



# On the Internet

- EvoNet: http://www.evonet.polytechnique.fr/
- www.genetic-programming.com
- GALib http://lancet.mit.edu/ga/
- http://www.aic.nrl.navy.mil/galist/
- www.isgec.org