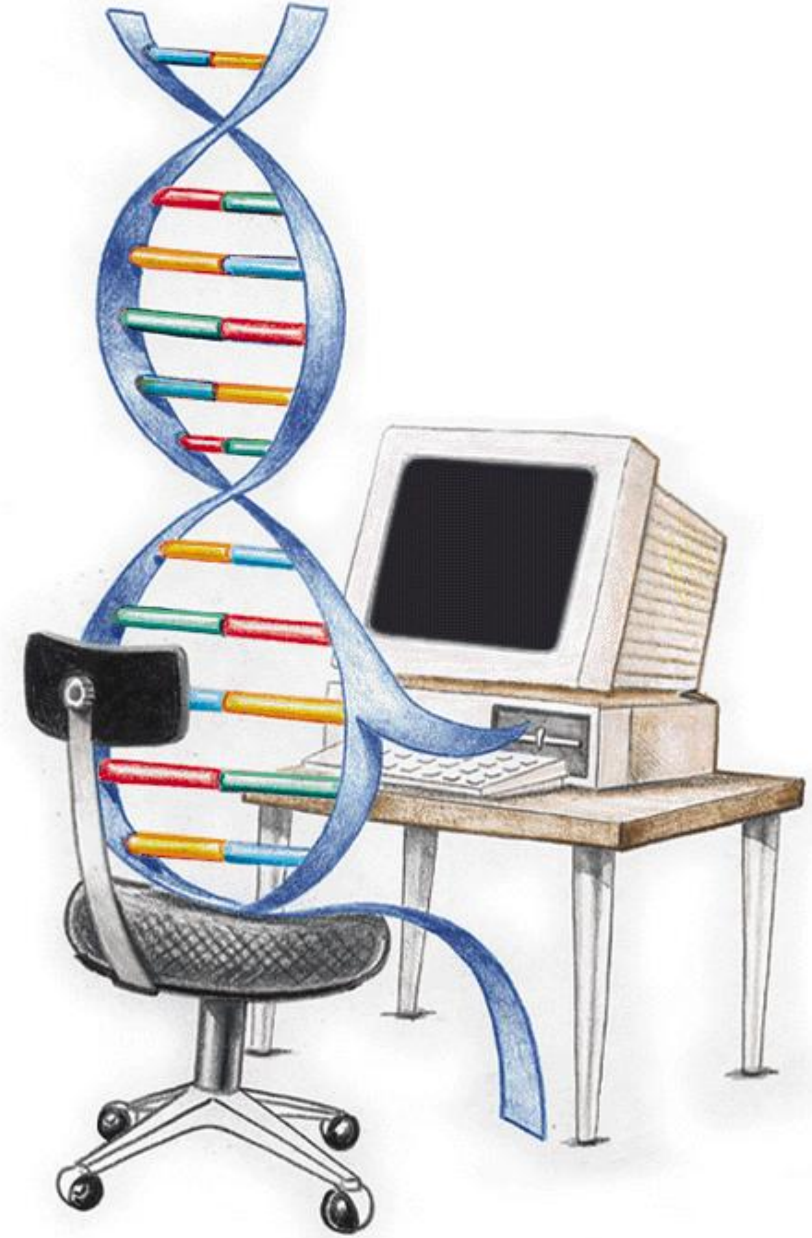
University of Ljubljana, Faculty of Computer and Information Science

Nature inspired computing



Prof Dr Marko Robnik Šikonja
Intelligent Systems
Edition 2024

Contents

- Introduction to evolutionary computation
- Genetic algorithms
- Genetic algorithms and automatic code generation

Evolutionary and natural computation

- Many engineering and computational ideas from nature work fantastically!
- Evolution as an algorithm
- Abstraction of the idea:
  - progress, adaptation - learning, optimization
- Survival of the fittest - competition of agents, programs, solutions
- Populations – parallelization
- (Over)specialization – local extremes
- Neuro-evolution, evolution of robots, evolution of novelty
- revival of interest

Template of evolutionary program

generate a population of agents (objects, data structures)

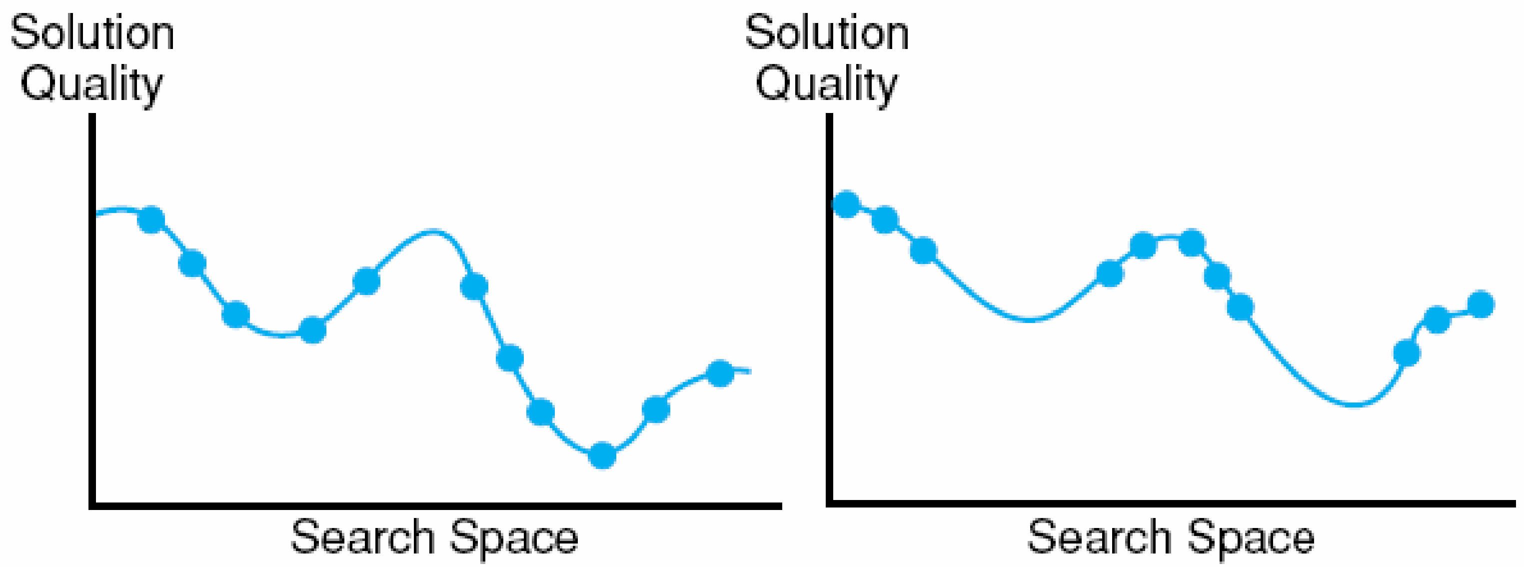
do {

compute fitness (quality) of the agents
select candidates for the reproduction using fitness
create new agents by combining the candidates
replace old agents with new ones

} while (not satisfied)

- immensely general -> many variants

A result of successful evolutionary program



a. The beginning search space

b. The search space after n generations

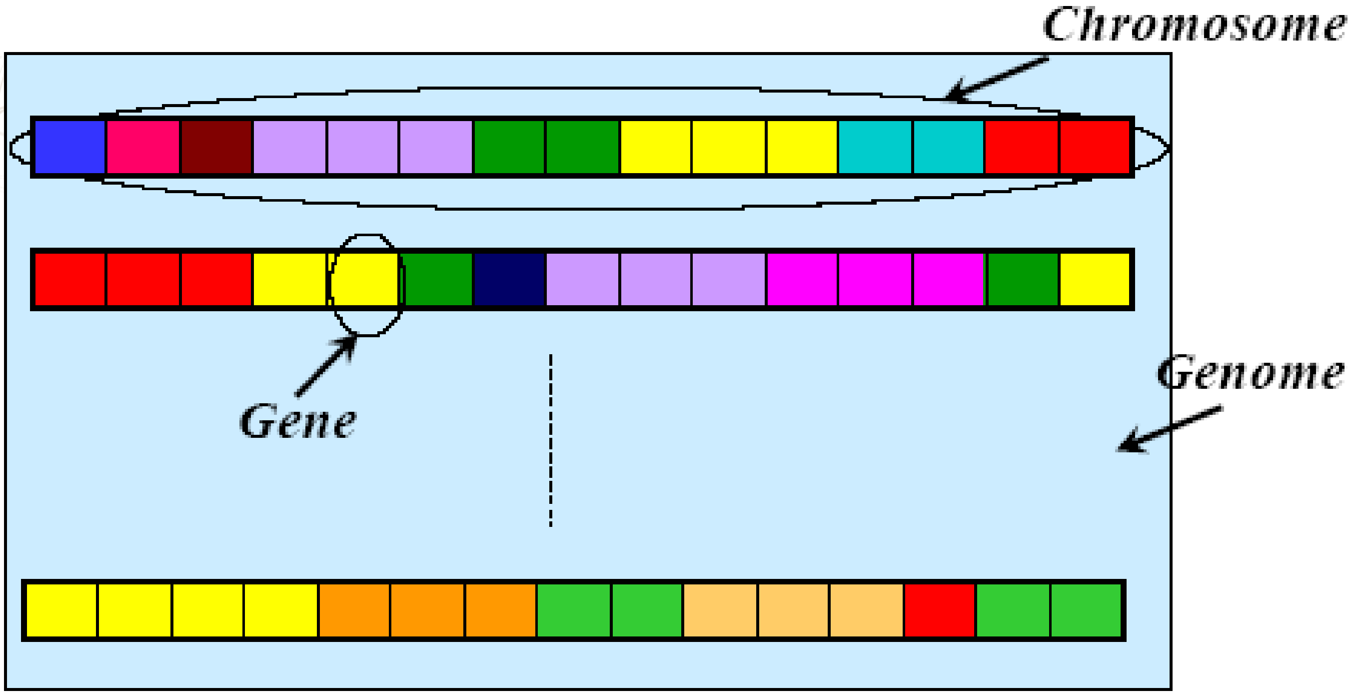
Main approaches

- Genetic algorithms
- Genetic programming
- Swarm methods (particles, ants, bees, …)
- Self-organized fields
- Differential evolution
- etc.

Genetic Algorithms - History

- Pioneered by John Holland in the 1970's
- Got popular in the late 1980's
- Based on ideas from Darwinian evolution
- Can be used to solve a variety of problems that are not easy to solve using other techniques

Chromosome, Genes and Genomes



Gene representation

- Bit vector
- Numeric vectors
- Strings
- Permutations
- Trees: functions, expressions, programs
- …

Crossover

- Single point/multipoint
- Shall preserve individual objects

Crossover: bit representation

Parents: 1101011100 0111000101

Children: 1101010101 0111001100

Crossover: vector representation

Simplest form

Parents: (6.13, 4.89, 17.6, 8.2) (5.3, 22.9, 28.0, 3.9)

Children: (6.13 , 22.9, 28.0, 3.9) (5.3, 4.89, 17.6, 8.2)

In reality: linear combination of parents

Linear crossover

- The linear crossover simply takes a linear combination of the two individuals.
- Let $x = (x_1, \ldots x_N)$ and $y = (y_1, \ldots y_N)$
- Select α in (0, 1)
- The results of the crossover is $\alpha x + (1 - \alpha) y$.
- Possible variation: choose a different α for each position.

Linear crossover example

- Let $\alpha = 0.75$ and we have this two individuals:

  $A = (5, 1, 2, 10)$ and $B = (2, 8, 4, 5)$

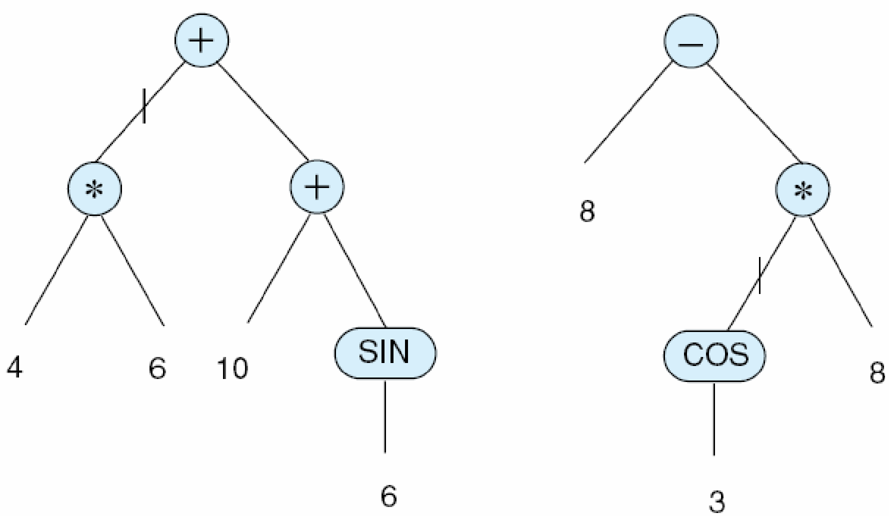
- then the result of the crossover is:

$(3.75 + 0.5, 0.75 + 2, 1.5 + 1, 7.5 + 1.25) = (4.25, 2.75, 2.5, 8.75)$

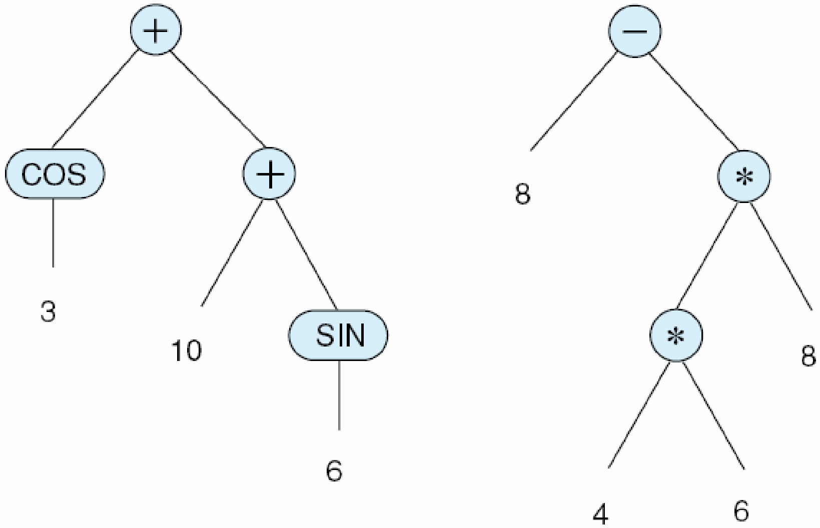
- If we use the variation and we have $\alpha = (0.5, 0.25, 0.75, 0.5)$, the result is:

$(2.5 + 1, 0.25 + 6, 1.5 + 1, 5 + 2.5) = (3.5, 6.25, 2.5, 7.5)$

Crossover: trees



a. b.



a. b.

Permutations: travelling salesman problem

- 9 cities: 1,2 ..9
- bit representation using 4 bits?
  - 0001 0010 0011 0100 0101 0110 0111 1000 1001
  - crossover would give invalid genes
- permutation and ordered crossover
  - keep (part of) sequences
  - use the sequence from second cut, keep already existing

1 9 2 | 4 6 5 7 | 8 3 → x x x | 4 6 5 7 | x x ↘ 2 3 9 | 4 6 5 7 | 1 8

4 5 9 | 1 8 7 6 | 2 3 → x x x | 1 8 7 6 | x x ↗ 3 9 2 | 1 8 7 6 | 4 5

A demo: Eaters

- Plant eaters are simple organisms, moving around in a simulated world and eating plants
- Fitness function: number of plants eaten
- An eater sees one square in front of its pointed end; it sees 4 possible things: another eater, plant, empty square or the wall
- Actions: move forward, move backward, turn left, turn right
- It is not allowed to move into the wall or another eater
- Internal state: number between 0 and 15
- The behavior is determined by the 64 rules encoded in its chromosome; one rule for each of 16 states x 4 observations; one rule is a pair (action, next state)
- The chromosome therefore consists of length 64 x (4+2) bits = 384 bits
- Crossover and mutation

Gray coding of binary numbers

- Keeping similarity

Binary	Gray
0000	0000
0001	0001
0010	0011
0011	0010
0100	0110
0101	0111
0110	0101
0111	0100
1000	1100
1001	1101
1010	1111
1011	1110
1100	1010
1101	1011
1110	1001
1111	1000

Adaptive crossover

- Different evolution phases
- Crossover templates
- 0 – first parent, 1 second parent
- Possibly different dynamics of template

	Gene	Template
Parent 1	1.2 3.4 5.6 4.5 7.9 6.8	010101
Parent 2	4.7 2.3 1.6 3.2 6.4 7.7	011100
Child 1	1.2 2.3 5.6 3.2 7.9 7.7	010100
Child 2	4.7 3.4 1.6 4.5 6.4 6.8	011101

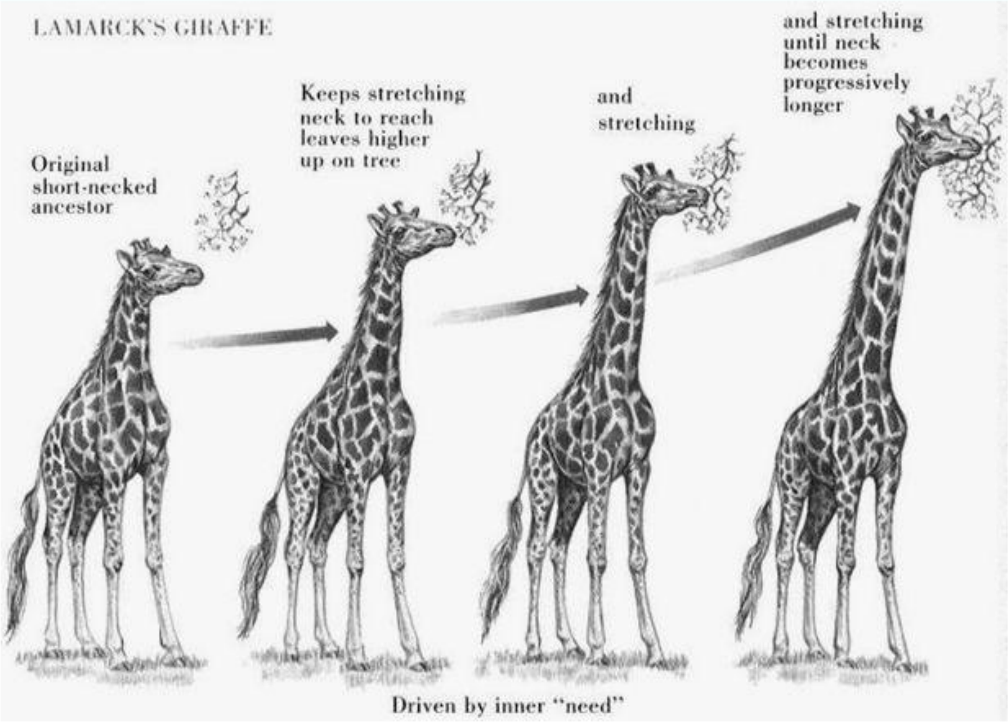
Mutation

- Adding new information
- Binary representation:
  0111001100 --> 0011001100
- Single point/multipoint
- Random search?
- Lamarckian (searching for locally best mutation)

Lamarckianism

Lamarckism is the hypothesis that an organism can pass on characteristics that it has acquired through use or disuse during its lifetime to its offspring.

An Early Proposal of Evolution: Theory of Acquired Characteristics



Jean Baptiste Lamarck (~ 1800) : Theory of Acquired Characteristics

- Use and disuse alter shape and form in an animal
- Changes wrought by use and disuse are heritable
- Explained how a horse-like animal evolved into a giraffe

Gaussian mutation

- When mutating one gene, selecting the new value by choosing uniformly among all the possible values is not the best choice (empirically).
- The mutation selects a position in the vector of floats and mutates it by adding a Gaussian error: a value extracted according to a normal distribution with the mean 0 and certain variance depending on the problem.

Template of evolutionary program

generate a population of agents (objects, data structures)

do {

compute fitness (quality) of the agents
select candidates for the reproduction using fitness
create new agents by combining the candidates
replace old agents with new ones

} while (not satisfied)

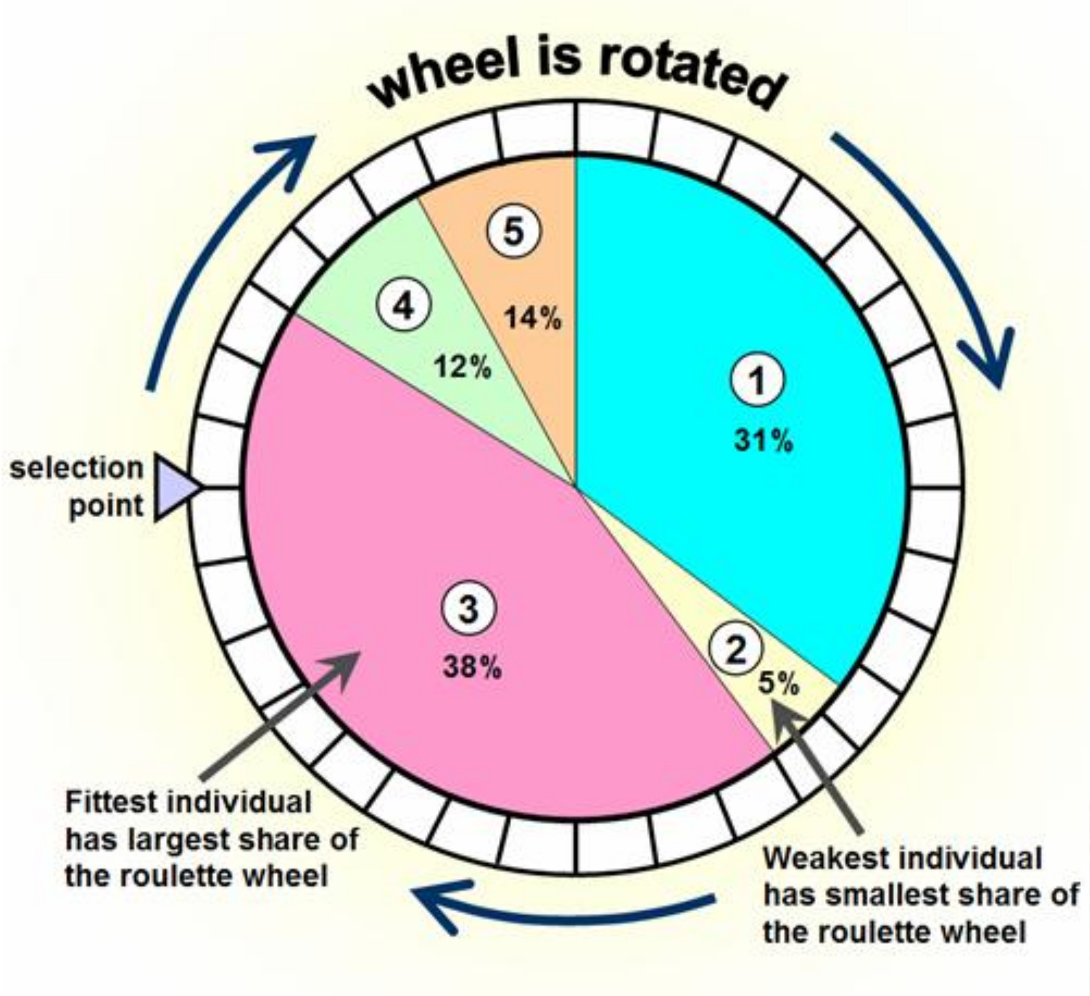
- immensely general -> many variants

Evolutional model - who will reproduce

- Keeping the good
- Prevent premature convergence
- Assure heterogeneity of population

Selection

- Proportional
- Rank proportional
- Tournament
- Single tournament
- Stochastic universal sampling



Tournament selection

1. set t=size of the tournament,
   p=probability of a choice
2. randomly sample t agents from population forming a tournament
3. select the best with probability p
4. select second best with probability p(1-p)
5. select third best with probability $p(1-p)^2$
6. …

Replacement

- All
- According to the fitness (roulette, rang, tournament, randomly)
- Elitism (keep a portion of the best)
- Local elitism (children replace parents if they are better)

Single tournament selection

1. randomly split the population into small groups
2. apply crossover to two best agents from each group; their offspring replace two worst agents from the group

- advantage: in groups of size *g* the best g-2 progress to next generation (we do not use good agents, maximal quality does not decrease)
- no matter the quality even the best agents have no more than two offspring (we do not loose population diversity)
- computational load?

Population size

- small, large?

Niche specialization

- evolutionary niches are generally undesired
- punish too similar agents

$f'_i = f_i / q(i)$

$$q(i) = \begin{cases} 1 & ; \text{sim}(i) <= 4, \\ \text{sim}(i)/4 & ; \text{otherwise} \end{cases},$$

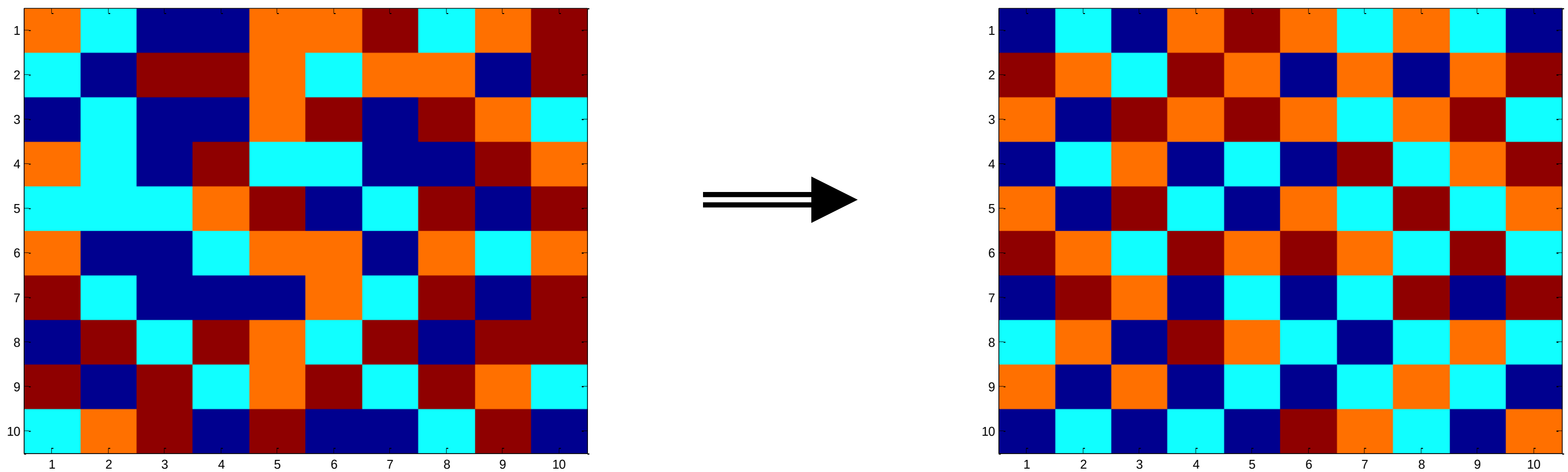
where sim(i) is the number of very similar agents to agent i

Stopping criteria

- number of generations, track progress, availability of computational resources, leaderboard mutability heuristics, etc.

Checkboard example

- We are given an ***n*** by ***n*** checkboard in which every field can have a different colour from a set of four colors.
- Goal is to achieve a checkboard in a way that there are no neighbours with the same color (not diagonal)

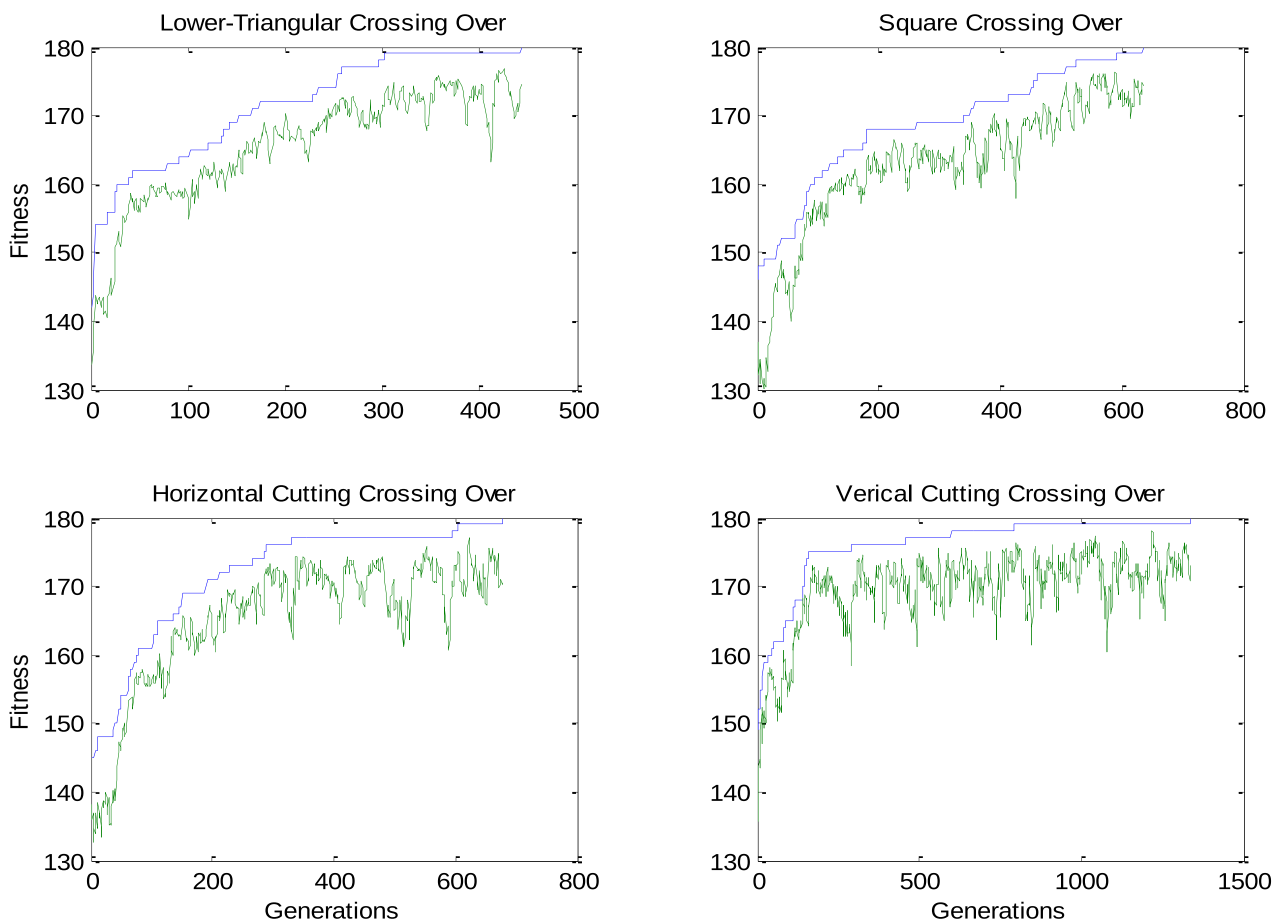


Checkboard example Cont'd

- Chromosomes represent the way the checkboard is colored.
- Chromosomes are not represented by bitstrings but by **bitmatrices**
- The bits in the bitmatrix can have one of the four values 0, 1, 2 or 3, depending on the color.
- Crossover involves matrix manipulation instead of point wise operating.
- Crossover can combine the parential matrices in a horizontal, vertical, triangular or square way.
- Mutation remains bitwise - changing bits
- Fitness function: check 2n(n-1) violations

Checkboard example Cont'd

- Fitness curves for different cross-over rules:



Why genetic algorithms work?

- building blocks hypothesis
- ... is controversial (mutations)
- sampling based hypothesis

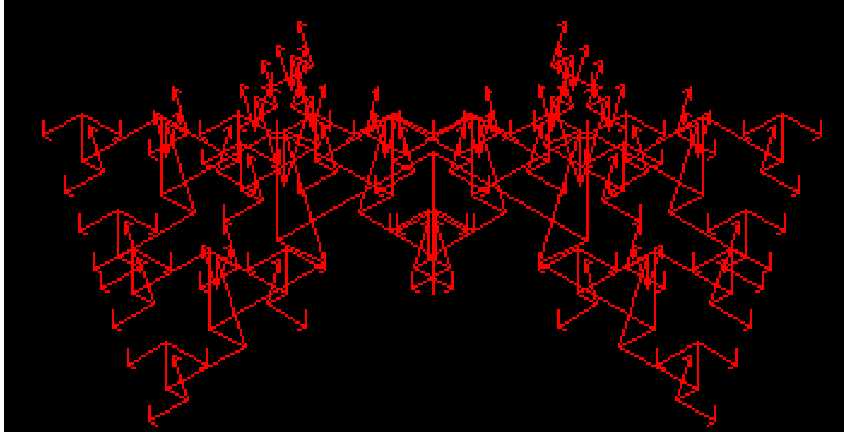
Parameters of GA

- Encoding (into fixed length strings)
- Length of the strings;
- Size of the population;
- Selection method;
- Probability of performing crossover (p_c);
- Probability of performing mutation (p_m);
- Termination criteria (e.g., a number of generations, a leaderboard mutability, a target fitness).

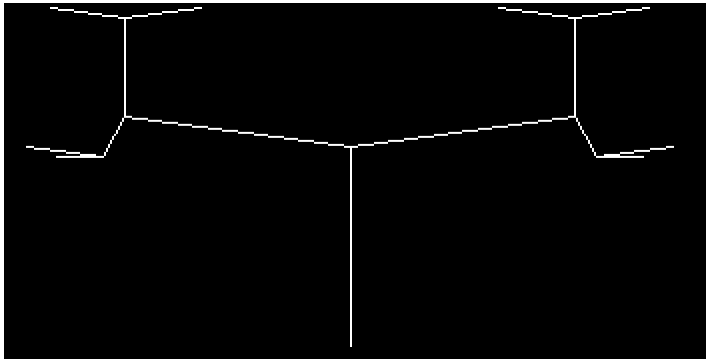
Usual settings of GA parameters

- Population size: from 20–50 to a few thousands individuals;
- Crossover probability: high (around 0.9);
- Mutation probability: low (below 0.1).

Demo: find genome of a biomorph



- A biomorph is a graphic configuration generated from nine genes.
- The first eight genes each encode a length and a direction.
- The ninth gene encodes the depth of branching.
- Each gene is encoded with five bits.
  - The four first bits represent the value, the fifth its sign.
  - Each gene can get a value from -15 to +15.
  - value of gen nine is limited to 2-9.



- There are : 8 (number of possible depths) x 2^{40} (the 8 * 5 =40 bits encoding basic genes) = 8.8 x10^{12} possible biomorphs. If we were able to test 1000 genomes every second, we would need about 280 years to complete the whole search.
- At the beginning, the drawing algorithm being known, we get the image of a biomorph. The only informations directly measurable are the positions of branching points and their number. The basic algorithm simulates the collecting of these informations.
- Fitness function: the distance of the generated biomorph from the target one.

Applications

- optimization
- scheduling
- bioinformatics,
- machine learning
- planning
- multicriteria optimization

Where to use evolutionary algorithms?

- Many local extremes
- Just fitness, without derivations
- No specialized methods
- Multiobjective optimization
- Robustness
- Combined approaches

Multiobjective optimization

- Fitness function with several objectives
- Cost, energy, environmental impact, social acceptability, human friendliness
- $\min F(x)=\min (f_1(x), f_2(x), \ldots, f_n(x))$
- Pareto optimal solution: we cannot improve one criteria without getting worse on others
- GA: in reproduction, use all criteria

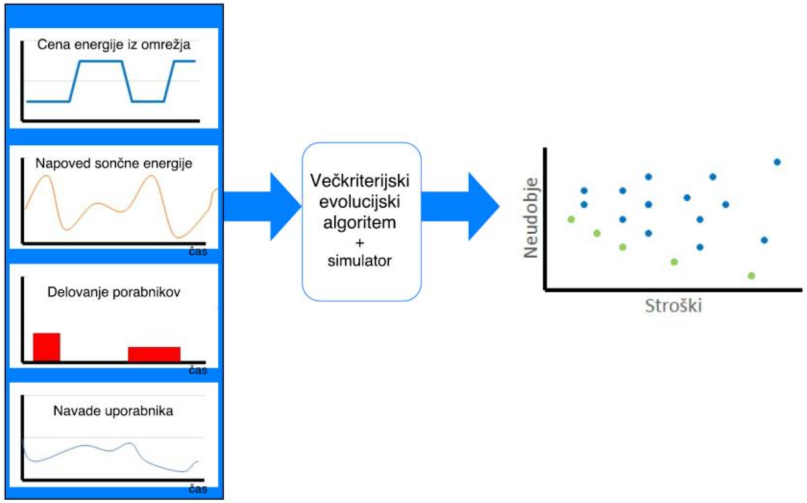
An example:
smart buildings

- simple scenario: heater, accumulator, solar panels, electricity from grid
- criteria: price, comfort of users (as the difference in temperature to the desired one)
- chromosome: shall encode schedule of charging and discharging the battery, heating on/off
- operational time is discretized to 15min intervals

Control problem for smart buildings

Parameters:

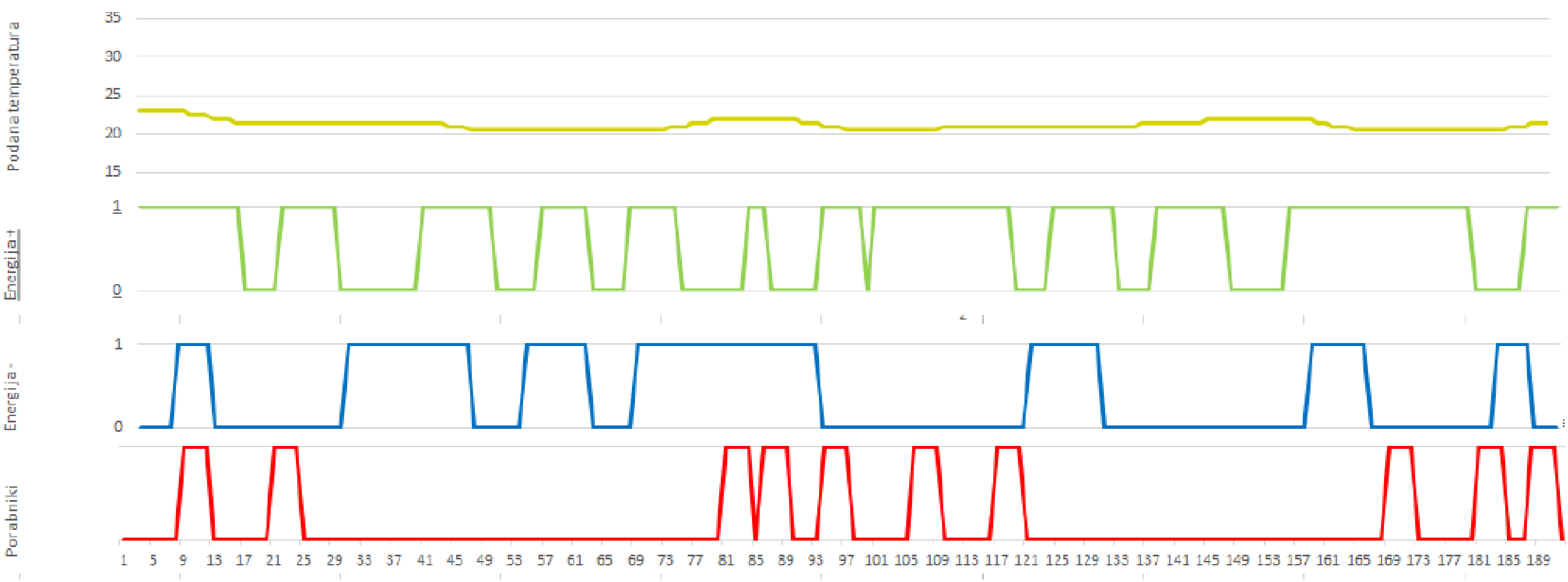
- the price of energy from the grid varies during the day
- the prediction of solar activity
- schedule of heater and battey
- usual activities of a user



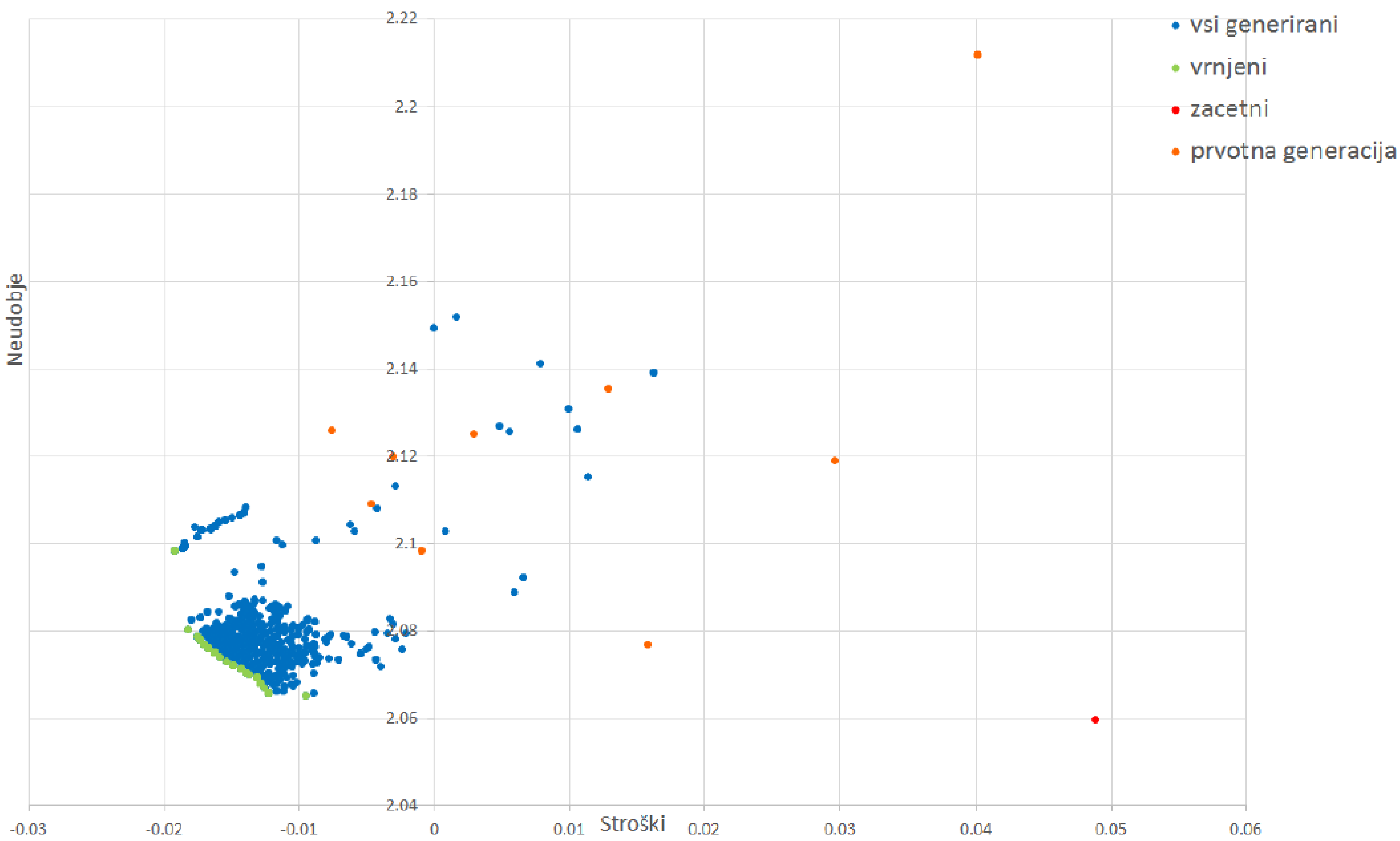
Smart building: structure of the chromosome

- temperature: for each interval we set the desired temperature between Tmin and Tmax interval
- battery+: if photovoltaic panels produce enough energy we set: 1 charging, 0 no charging
- battery-: if photovoltaic panels do not produce enough energy, we set: 1 battery shall discharge, 0 battery is not used
- appliances: each has its schedule when it is used (1) and when it is off (0)

Example of schedule



Example of solutions and optimal front



Toolboxes and libraries

- CIlib – computational intelligence library
- EO (C++) - evolutionary computation library
- ECF- Evolutionary Computation Framework (C++)
- ECJ, EvA2, JAGA (Java)
- R: Rfreak, ppso, numDeriv, etc
- Matlab

Strengths and weaknesses

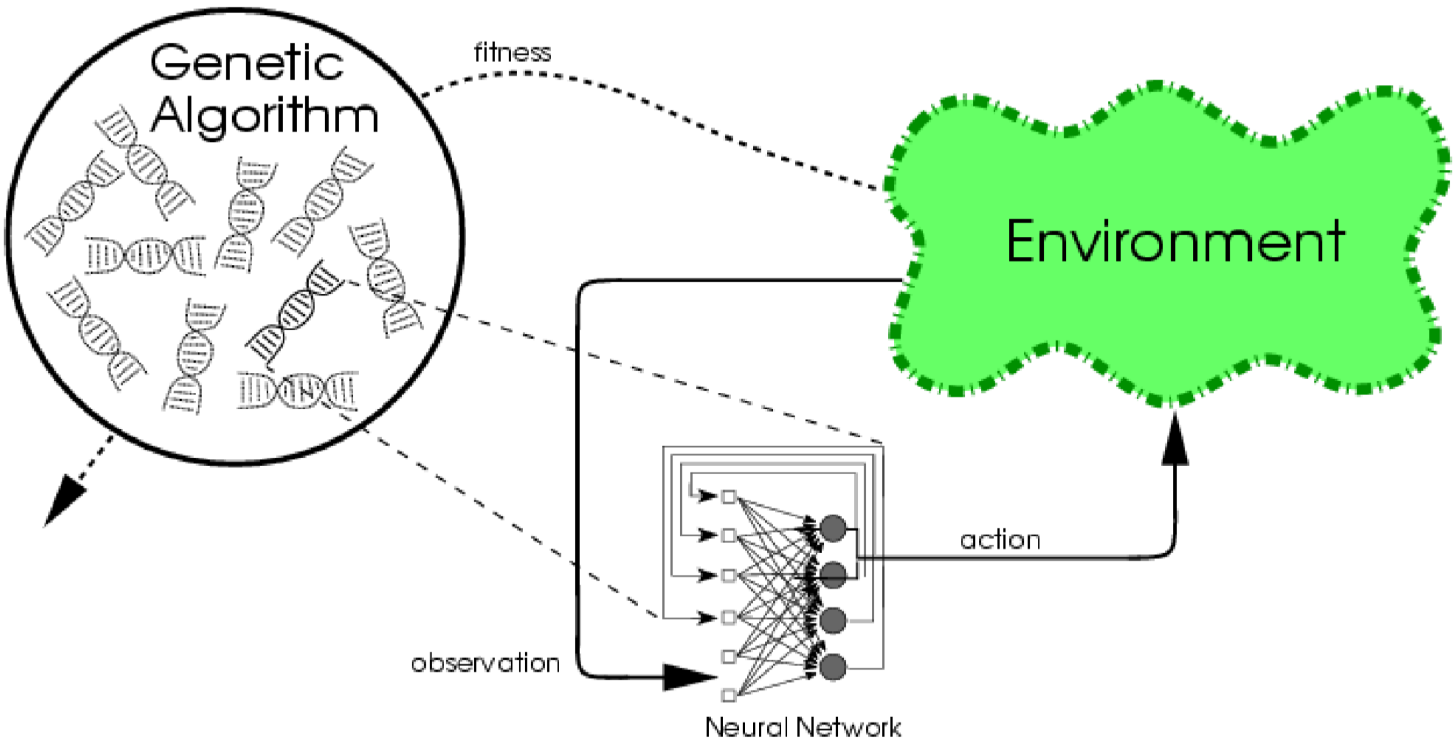
- robust, adaptable, general
- requires only weak knowledge of the problem (fitness function and representation of genes)
- several alternative solutions
- hybridization and parallelization
- faster and less memory than (exhaustive, random) search
- little effort to try

- suboptimal solutions
- possibly many parameters
- may be computationally expensive

- no-free-lunch theorem

Neuroevolution: evolving neural networks

- Evolving neurons and/or topologies



Neuroevolution

- Evolving neurons: not really necessary but attempted
- Evolving weights instead of backpropagation and gradient descent
- Evolving the architecture of neural network
  - For small nets, one uses a simple matrix representing which neuron connects which.
  - This matrix is, in turn, converted into the necessary 'genes', and various combinations of these are evolved.