

Genetic algorithms and hybrids



Contents

- ✱ Introduction to evolutionary computation
- ✱ Genetic algorithms
- ✱ Memetic algorithm

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

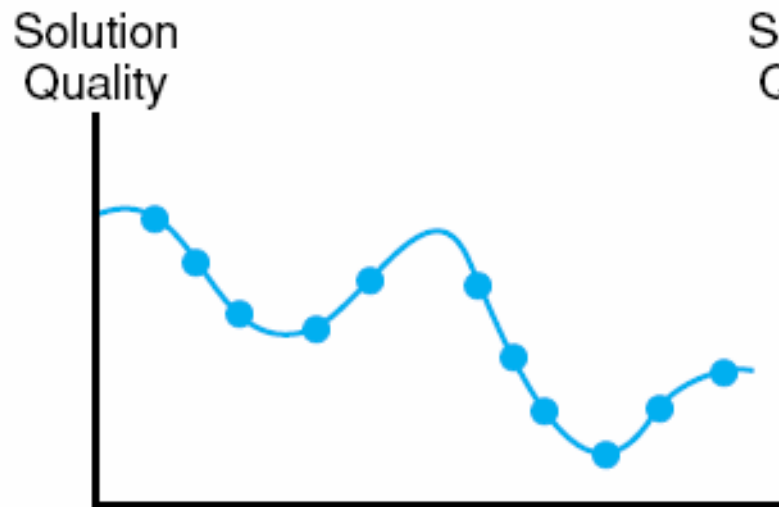
 create new agents by combining the candidates

 replace old agents with new ones

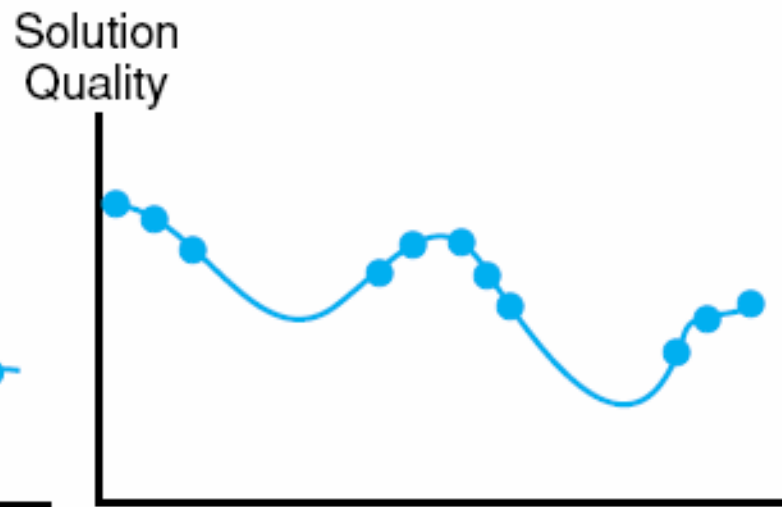
} while (not satisfied)

✱ immensely general -> many variants

A result of a successful evolutionary program



a. The beginning search space



b. The search space after n generations

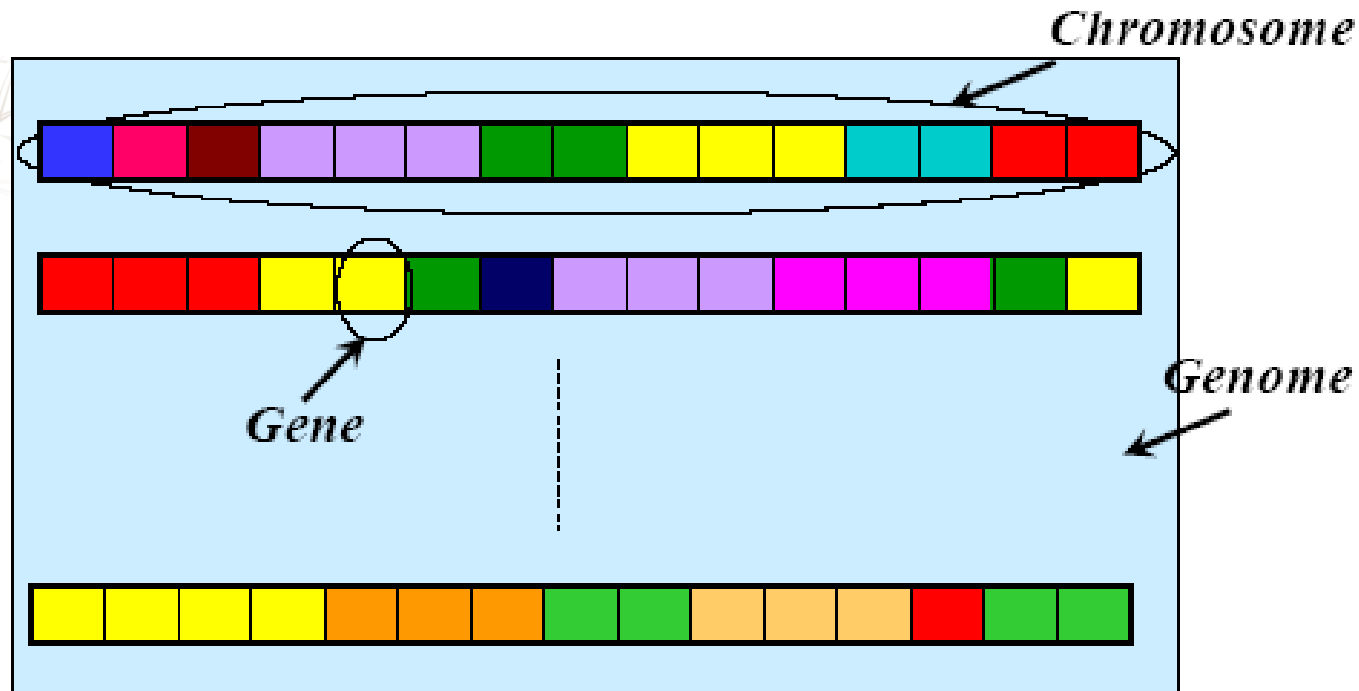
Main evolutionary approaches

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

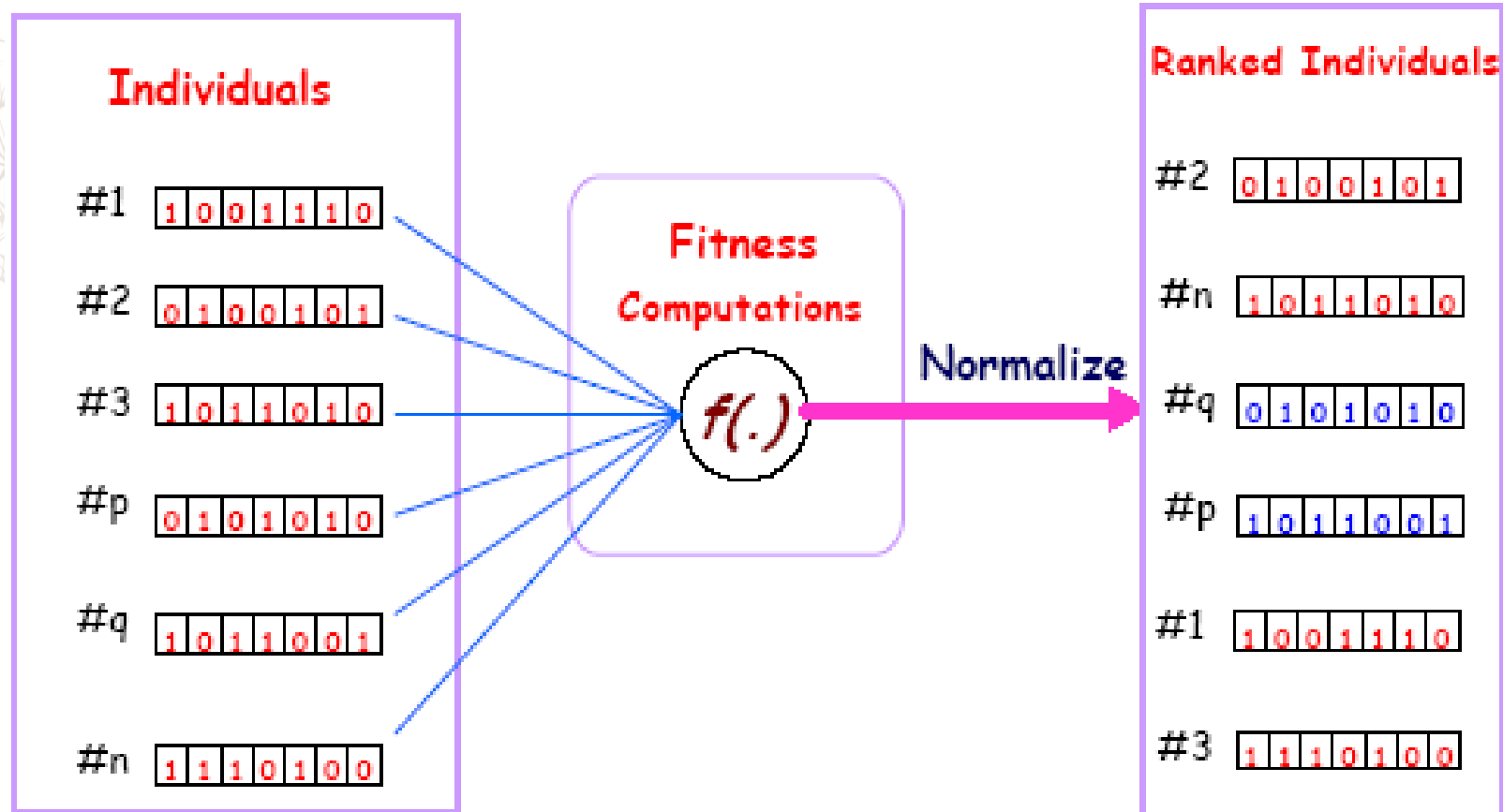
Genetic Algorithms - History

- ✱ 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



A fitness function



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** 011100**1100**

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 $x = (x_1, \dots, x_N)$ and $y = (y_1, \dots, y_N)$
- ✱ 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:

$$A = (5, 1, 2, 10) \text{ and } B = (2, 8, 4, 5)$$

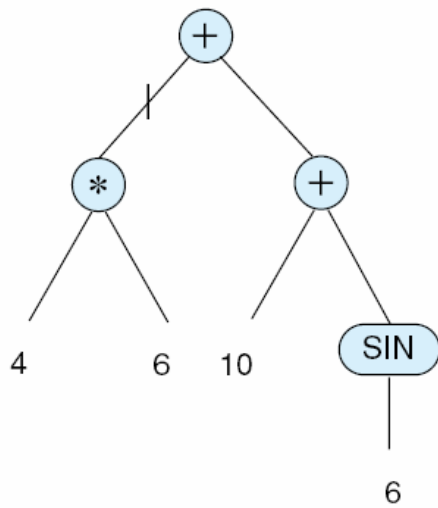
- ✱ Then the result of the crossover is:

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

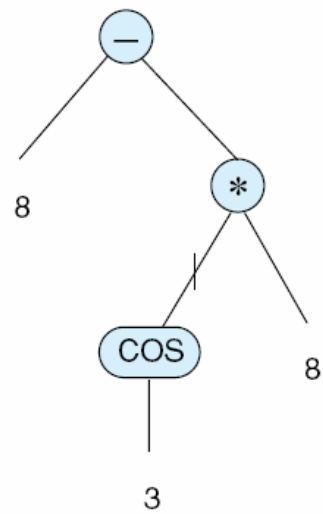
- ✱ If we use the variation and we have $\alpha = (0.5, 0.25, 0.75, 0.5)$, the result is:

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

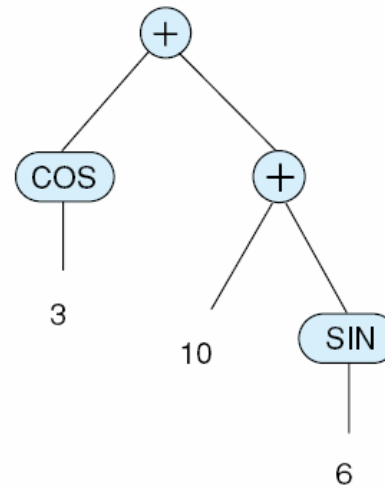
Crossover: trees



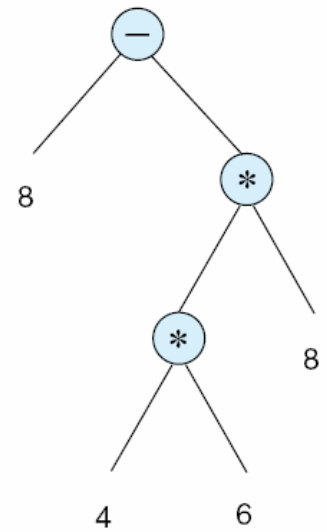
a.



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 $64 \times (4+2)$ bits = 384 bits
- ✿ Crossover and mutation

Mutation

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

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)

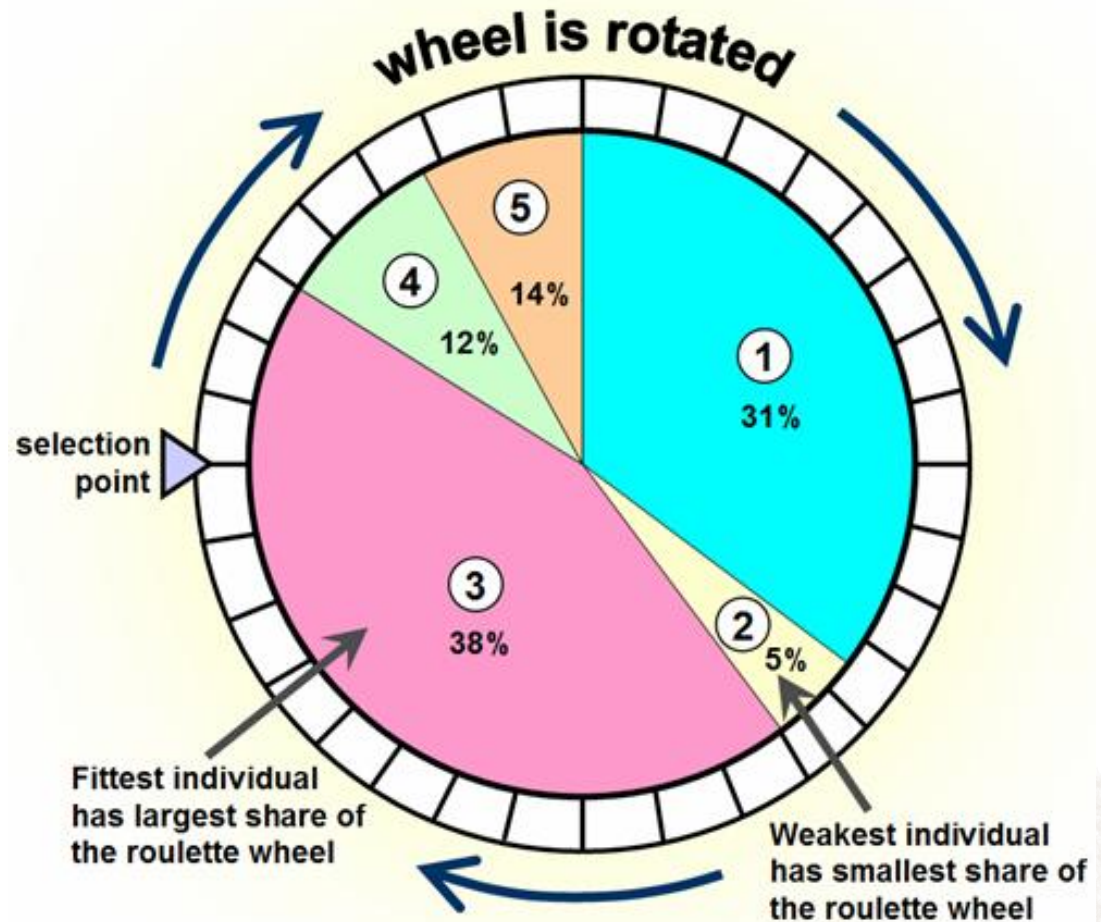
✱ immensely general -> many variants

Evolutional model - who will reproduce

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

Selection

- Proportional
- Rank proportional
- Tournament
- Single tournament



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 lose good agents, maximal quality does not decrease)
 - ✱ no matter the quality even the best agents have no more than two offspring (we do not lose population diversity)
 - ✱ computational load?

Population size

✱ small, large?



Niche specialization

- ✱ evolutionary niches are generally undesired
- ✱ punish too similar agents

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

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

Stopping criteria

- ✱ number of generations, track progress, availability of computational resources, etc.

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).

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
- ✱ $\min F(x) = \min (f_1(x), f_2(x), \dots, f_n(x))$
- ✱ Pareto optimal solution: we cannot improve one criteria without getting worse on others
- ✱ GA: in reproduction, use all criteria

An example: smart buildings

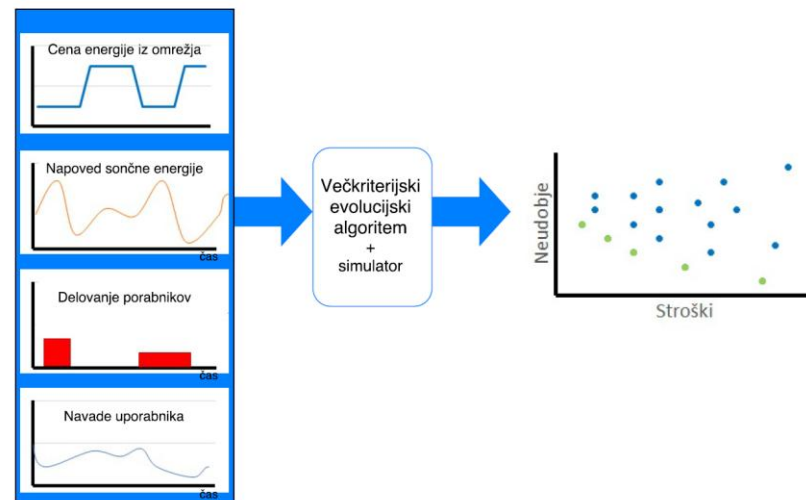


- ✱ simple scenario: heater, accumulator, solar panels, electricity from grid
- ✱ criteria: price, comfort of users (as the difference in temperature to the desired one)
- ✱ chromosome: shall encode schedule of charging and discharging the battery, heating on/off
- ✱ operational time is discretized to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

Control problem for smart buildings

Parameters:

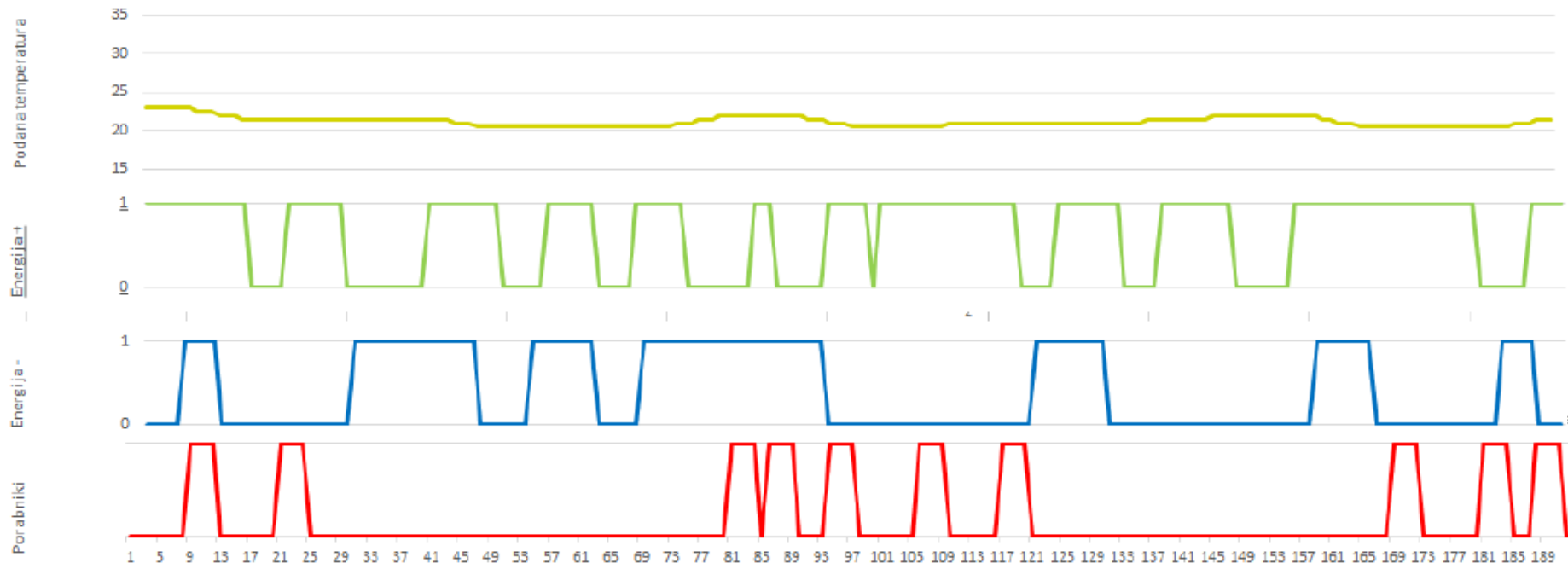
- the price of energy from the grid varies during the day
- the prediction of solar activity
- schedule of heater and battery
- usual activities of a user



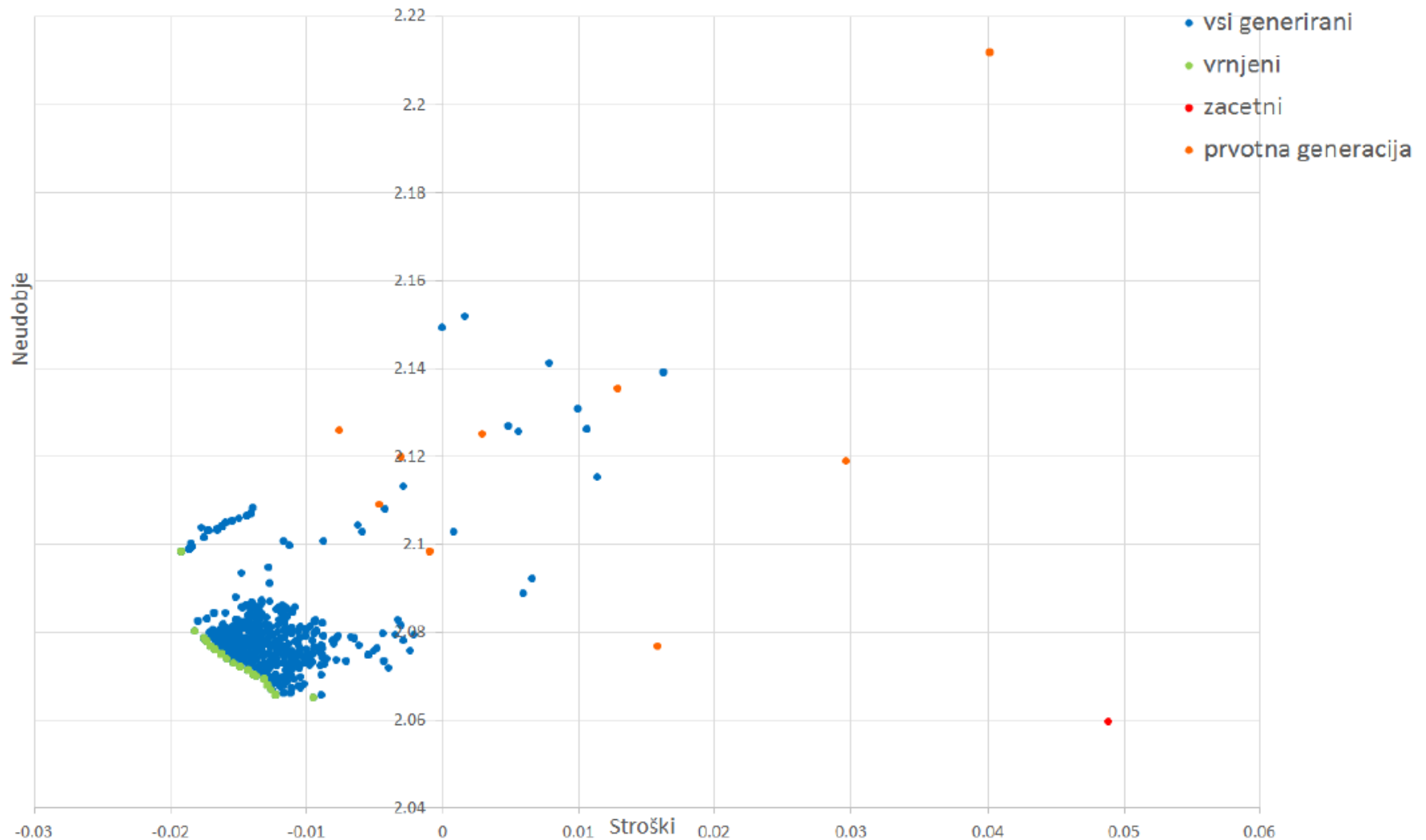
Smart building: structure of the chromosome

- ✿ temperature: for each interval we set the desired temperature between T_{min} and T_{max} 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



Pros and Cons of GA

☀ Pros

- ✂ Faster (and lower memory requirements) than searching a very large search space.
- ✂ Easy, in that if your candidate representation and fitness function are correct, a solution can be found without any explicit analytical work.

☀ Cons

- ✂ Randomized – not optimal or even complete.
- ✂ Can get stuck on local maxima, though crossover can help mitigate this.
- ✂ It can be hard to work out how best to represent a candidate as a bit string (or otherwise).

Genetic programming

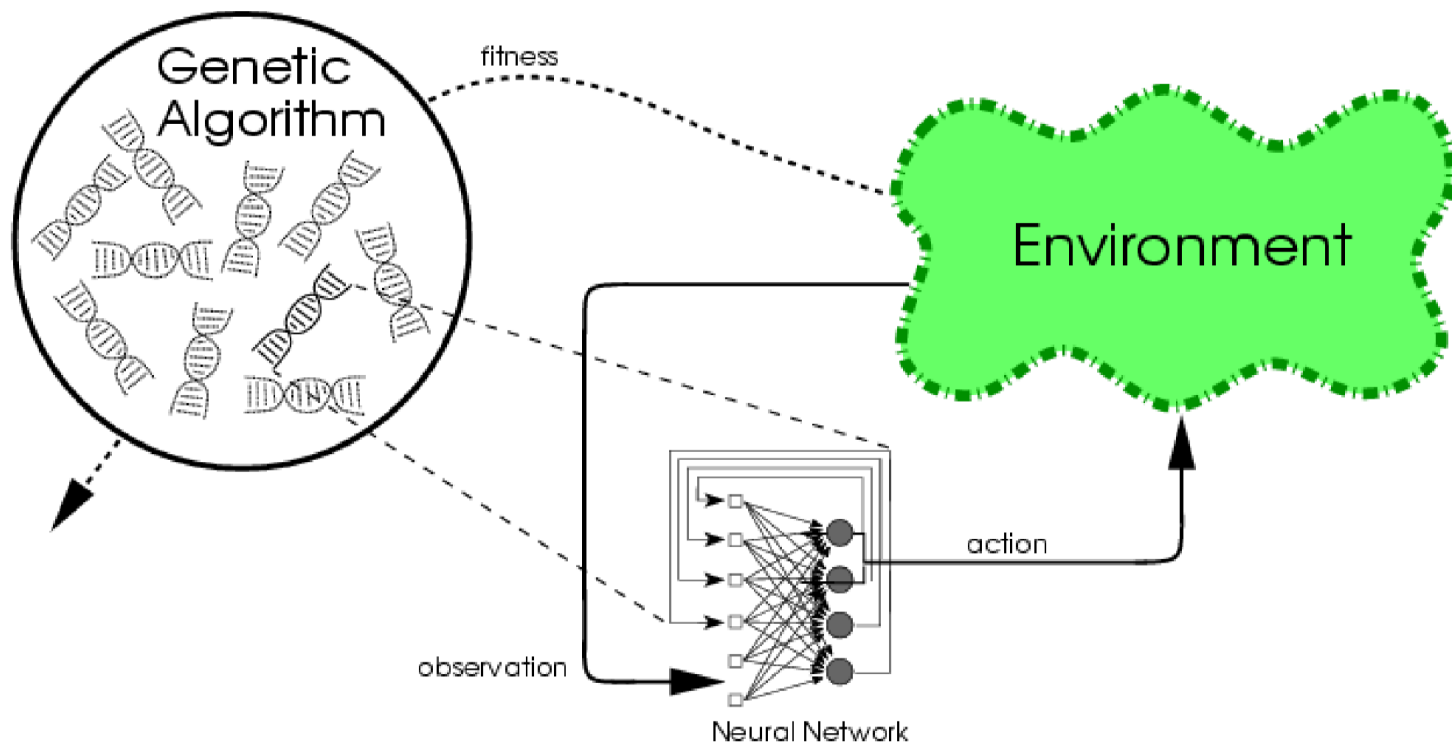
- ✱ Functions, programs, expression trees
- ✱ Keep the structures valid
- ✱ Tree crossover, type closure

GP quick overview

- ✿ Developed: USA in the 1990's
- ✿ Early names: J. Koza
- ✿ Typically applied to:
 - ✗ machine learning tasks (prediction, classification...)
 - ✗ controller design
 - ✗ function fitting
- ✿ Attributed features:
 - ✗ competes with neural nets and alike
 - ✗ needs huge populations (thousands)
 - ✗ slow
- ✿ Special:
 - ✗ non-linear chromosomes: trees, graphs
 - ✗ mutation possible but not necessary (disputed!)
- ✿ large potential, but so far did not deliver much

Neuroevolution: evolving neural networks

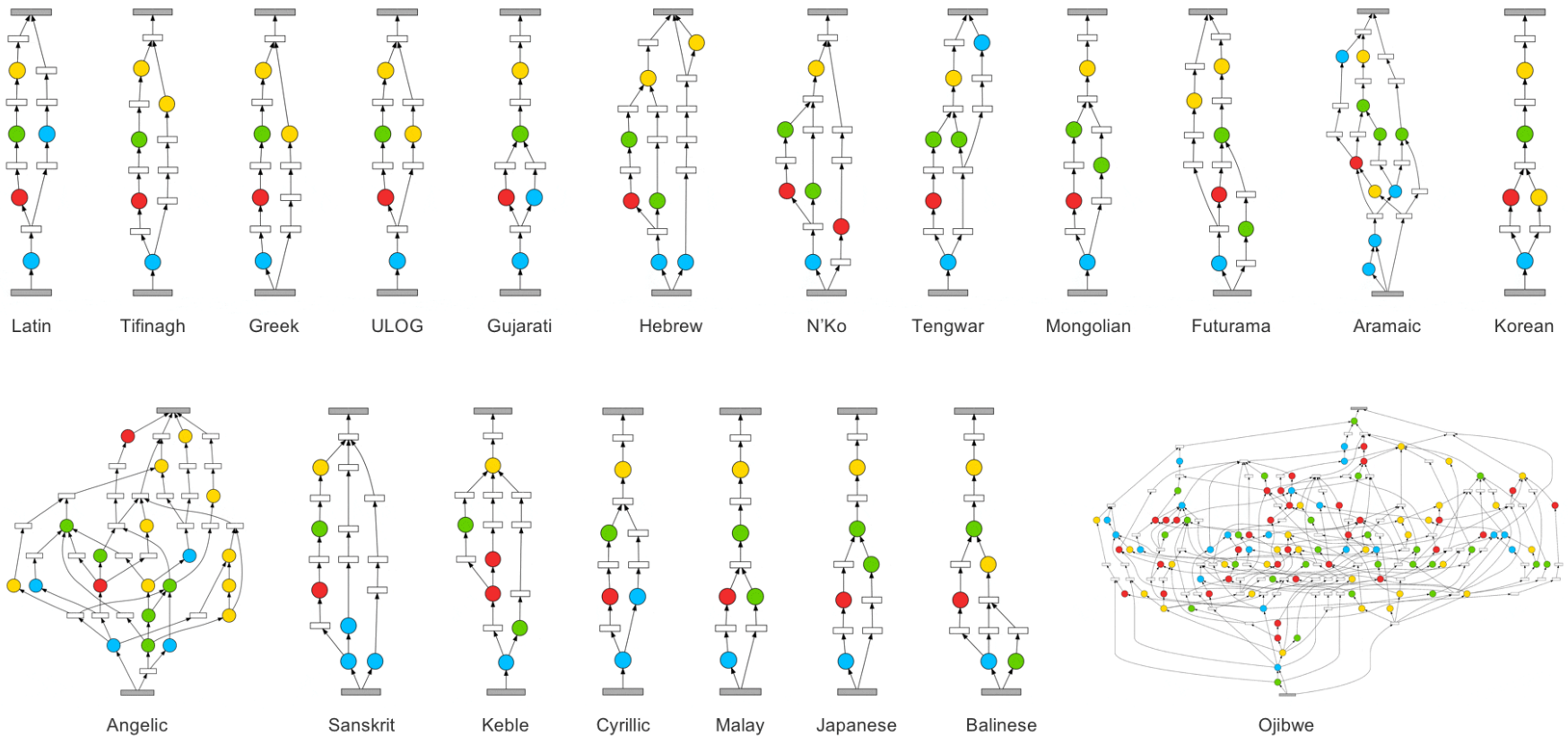
- Evolving neurons and/or topologies



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.

Example: multialphabet character recognition architectures



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)

Memetic algorithms

- ✱ An attempt to merge several ideas from combinatorial optimization

```
1  Procedure Population-Based-Search-Engine;  
2  begin  
3      Initialize pop using GenerateInitialPopulation();  
4      repeat  
5          newpop  $\leftarrow$  GenerateNewPopulation(pop);  
6          pop  $\leftarrow$  UpdatePopulation (pop, newpop);  
7          if pop has converged then  
8              pop  $\leftarrow$  RestartPopulation(pop);  
9          endif  
10     until TerminationCriterion() ;  
11 end
```

Memetic algorithms initialization

✱ Using local search

```
1  Procedure GenerateInitialPopulation;  
2  begin  
3      Initialize pop using EmptyPopulation();  
4      for  $j \leftarrow 1$  to popsize do  
5           $i \leftarrow$  GenerateRandomConfiguration();  
6           $i \leftarrow$  Local-Search-Engine ( $i$ );  
7          InsertInPopulation individual  $i$  to pop;  
8      endfor  
9      return pop;  
10 end
```

Memetic algorithms - restart

✱ elitism and local search

```
1  Procedure RestartPopulation (pop);
2  begin
3      Initialize newpop using EmptyPopulation();
4      #preserved  $\leftarrow$  popsize · %preserve;
5      for j  $\leftarrow$  1 to #preserved do
6          |   i  $\leftarrow$  ExtractBestFromPopulation(pop);
7          |   InsertInPopulation individual i to newpop;
8      endfor
9      for j  $\leftarrow$  #preserved + 1 to popsize do
10         |   i  $\leftarrow$  GenerateRandomConfiguration();
11         |   i  $\leftarrow$  Local-Search-Engine (i);
12         |   InsertInPopulation individual i to newpop;
13     endfor
14     return newpop;
15 end
```