# Genetic Programming

Lecturer: **Michal Bidlo**, bidlom@fit.vut.cz,
https://www.fit.vut.cz/person/bidlom/
Faculty of Information Technology
Brno University of Technology

Based on the study material of Lukáš Sekanina, sekanina@fit.vutbr.cz

For BISSIT 2022 adapted by M. Bidlo, July 22, 2022.

# Genetic programing (GP)

GP is a machine learning method that can automatically design

- computer programs (e.g., classifiers, predictors, neural networks, …), but also
- other structures that can be represented in a computer (such as electronic circuits, molecules, antennas, artefacts …)

without requiring the user to know or specify the form or structure of the solution in advance.

GP belongs to the class of Evolutionary Algorithms that are inspired in Darwin's theory of evolution and Neo-Darwinism.

# Human system design

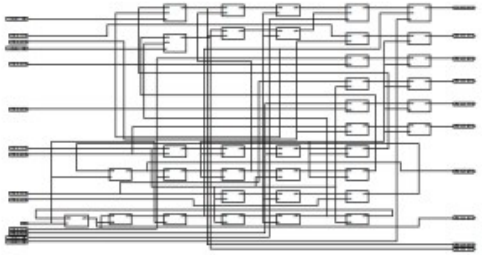
1 Specification

Requirements
Constraints
Training data
Similar solutions
etc.

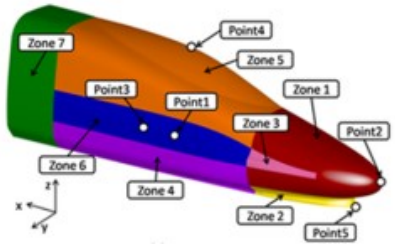
2 Design system

https://scitechdaily.com/researchers-discover-how-the-human-brain-separates-stores-and-retrieves-memories/

3 Solution



```
movf    NUM_2,w
call    SQR
movf    SQUARE+1,w
addwf   SUM+1,f
btfsc   STATUS,C
 incf   SUM,f
movf    SQUARE,w
addwf   SUM,f
```



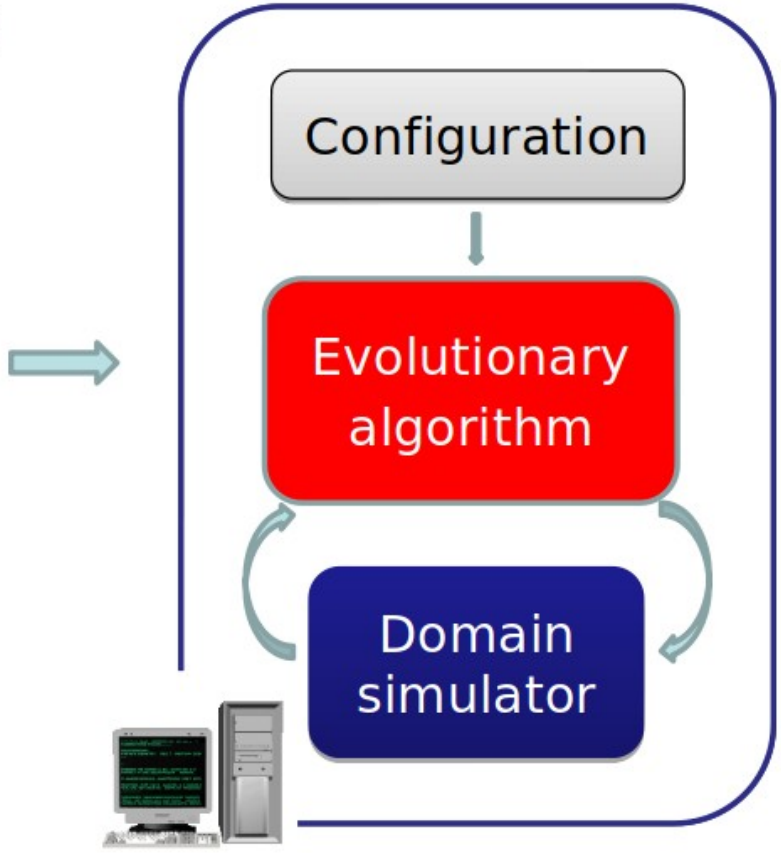
# Automated system design

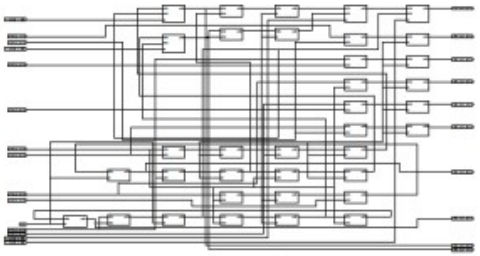
1 Specification

2 Design system

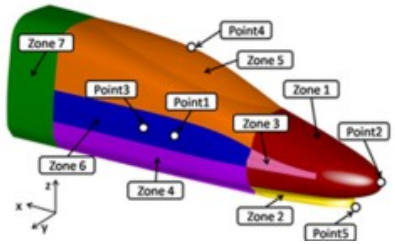
3 Solution

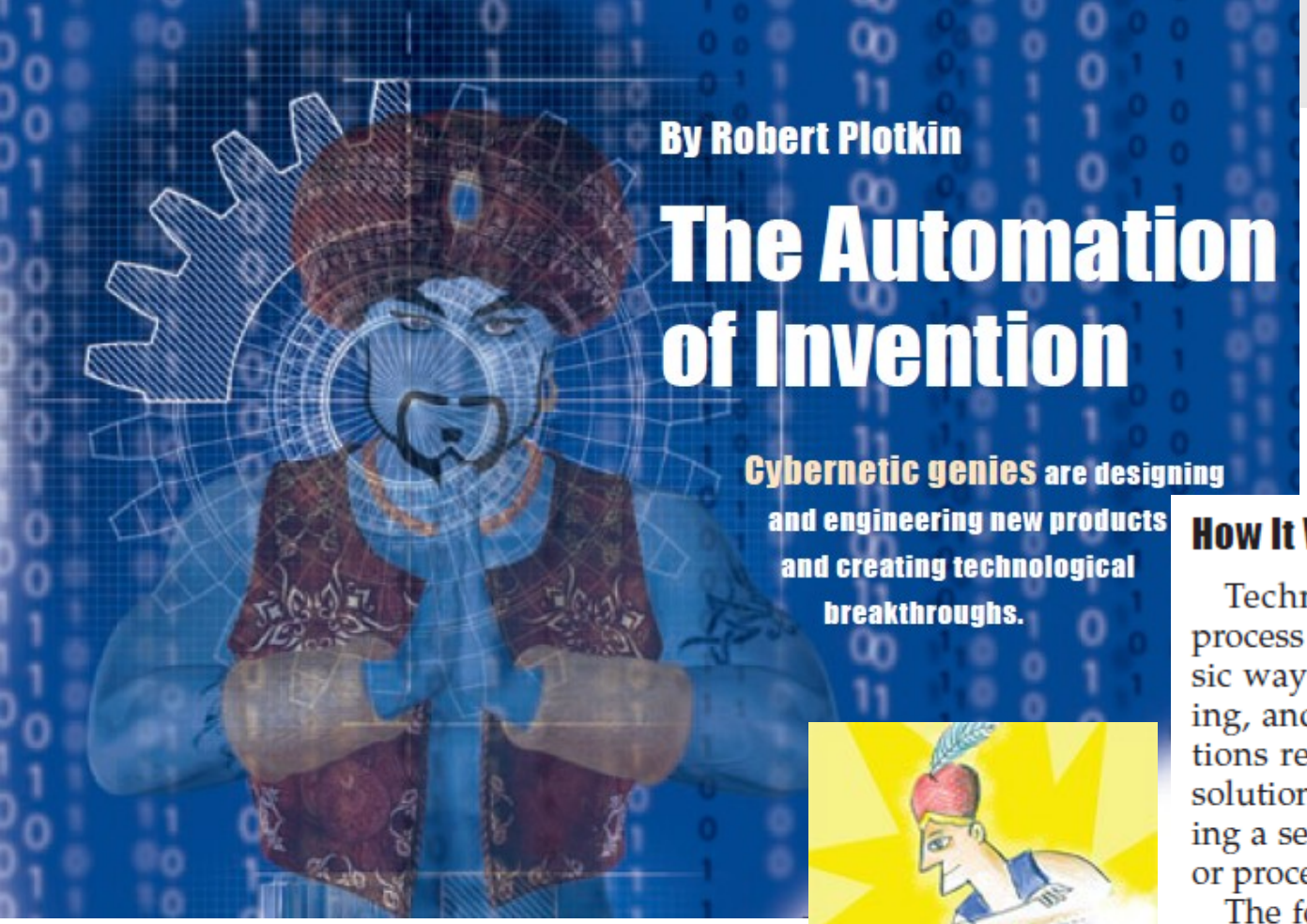
Requirements
Constraints
Training data
Similar solutions
etc.

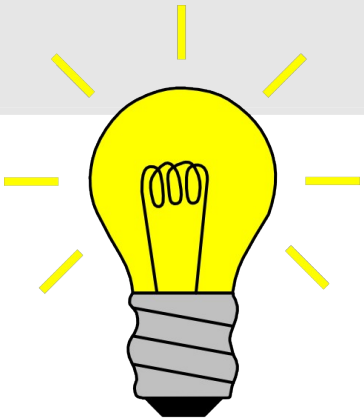


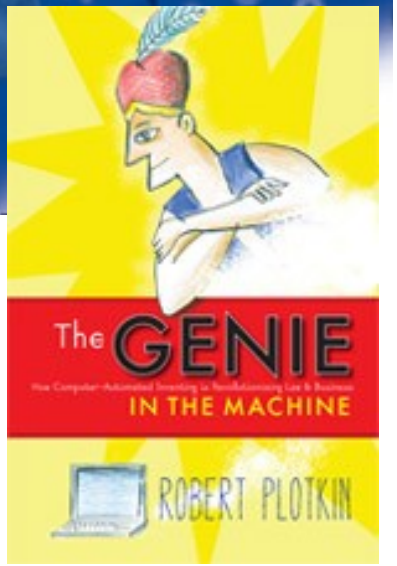



```
movf    NUM_2,w
call    SQR
movf    SQUARE+1,w
addwf   SUM+1,f
btfsc   STATUS,C
 incf   SUM,f
movf    SQUARE,w
addwf   SUM,f
```








## How It Works

Technologies that automate the process of inventing do so in two basic ways: (1) by generating, evaluating, and modifying potential inventions repeatedly until a satisfactory solution is found; and (2) by following a set of rules to design a product or process that achieves a goal.

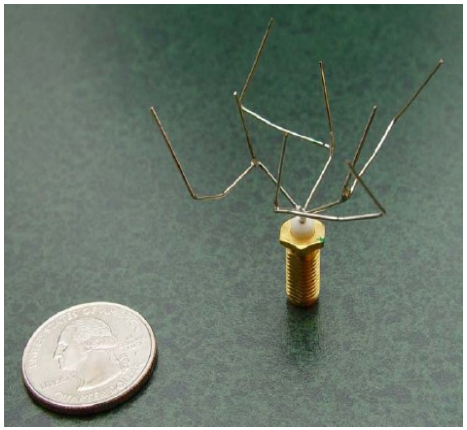
The former technique, which I call "inventing by searching," is often performed manually by human inventors. Thomas Edison invented an improved light bulb by hiring a team of experts to search the globe for better filament material. After testing more than 6,000 alternatives, he settled on carbonized bamboo.

World Future Society Special Reports 2010

# Automated design by GP

## Benefits

- New (unexpected) and useful designs and solutions can be obtained.
- Human effort is reduced.
- The design can be performed online and autonomously, in target applications such as adaptive, learning and self-repairing systems.



NASA AMES, ST5 Mission

## Problems

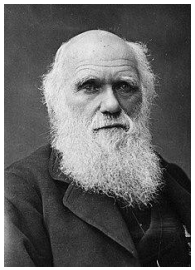
- Scalability
- Computation time (search method + simulator)
- New ethical issues

# Outline

- Introduction
- Evolutionary algorithms
- Genetic programming
- Cartesian genetic programming
- Applications from FIT
- Summary

# Darwinian Evolution

*All species of life have descended from common ancestors.*

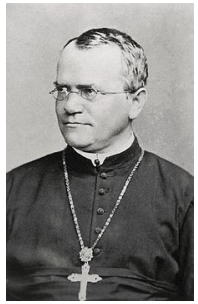


Charles Darwin
(1809 - 1882)

Four Postulates

1. Individuals within species are variable
2. Some of the variations are passed on to offspring
3. In every generation, more offspring are produced than can survive
4. The survival and reproduction of individuals are not random: The individuals who survive and go on to reproduce, or who reproduce the most, are those with the most favourable variations. They are naturally selected.

On the Origin of Species by Means of Natural Selection (Darwin, 1859)



Neo-Darwinism is generally used to describe the integration of Darwin's theory of evolution by natural selection with Mendel's theory of genetics.

Gregor J. Mendel
(1822 - 1884)
Brno

# Evolutionary algorithms (EA)

- **Evolutionary Algorithms** are inspired in Darwin's theory of evolution and Neo-Darwinism.
- The term Evolutionary Algorithm (EA) covers various *search* algorithms that have the following common features:
  - There is a population (a set) of individuals (so-called candidate solutions).
  - Each candidate solution gets a fitness value by evaluating its quality by means of an objective function (or fitness function). **This function is application specific.**
  - In each iteration (a generation) of EA, new candidate solutions are created by
    - selection of parent individuals (**those with better fitness are preferred = natural selection**),
    - offspring are created from parents by crossover and mutation operators (this process is called reproduction).
  - The goal is to optimize (minimize or maximize) the fitness, i.e. to find a solution which fulfils the requirements of the fitness function as best as possible.
  - EA is a stochastic algorithm; nothing is guaranteed :(
- **Observe the analogy with biology**: the offspring may inherit the features of parents (= crossover), sometimes with small random changes (= mutation) → new features may emerge, the offspring **evolve** during many generations.

# When can EA be used?

- If it is possible to encode a candidate solution (so-called **phenotype**, e.g., a digital circuits or NN) into a suitable computer representation (so-called **genotype**, e.g. a bit string, a sequence of “real” numbers...) and
- If it is possible to evaluate the quality of each candidate solution using a fitness function (e.g. fitness score: 0%...worst – 100%...best).

- EA are typically used to solve hard optimization and design problems for which common methods do not provide sufficient solutions.

- Main branches of EA:
  - Evolutionary Programming - EP (Fogel ~1962)
  - Evolution Strategies – ES (Rechenberg and Schwefel ~1964)
  - Genetic Algorithms – GA (Holland ~1973)
  - Genetic Programming – GP (Cramer ~1985, Schmidhuber ~1987, **Koza, ~1989**)
  - and some others variants currently exist...

# Example 1: evolutionary design of a logic circuit

- We must know what we want to design, i.e. the function of the circuit – this is typically specified as a truth table, e.g.:

| a | b | c | d | y |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 1 |
| 0 | 0 | 1 | 0 | 1 |
| 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 1 | 1 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 0 | 1 | 1 | 1 |
| 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 |

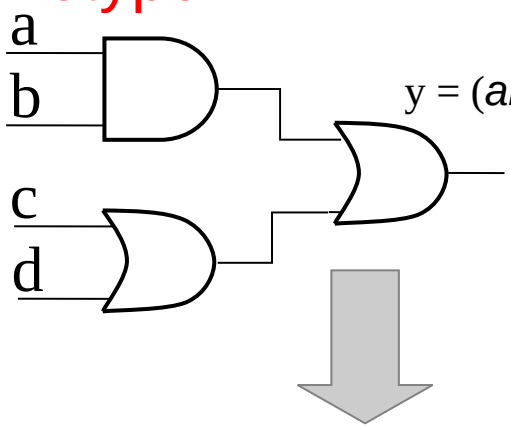
- EA then performs a search in the space of possible candidate solutions using a (small) subset of candidate solutions and bio-inspired operators in order to maximize the fitness score.

Evaluation of a candidate sol.:

genotype (a *chromosome*)

0011001111001010100011000101

phenotype




| a | b | c | d | y |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 1 |
| 0 | 0 | 1 | 0 | 1 |
| 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 1 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 0 | 1 | 1 | 1 |
| 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 |

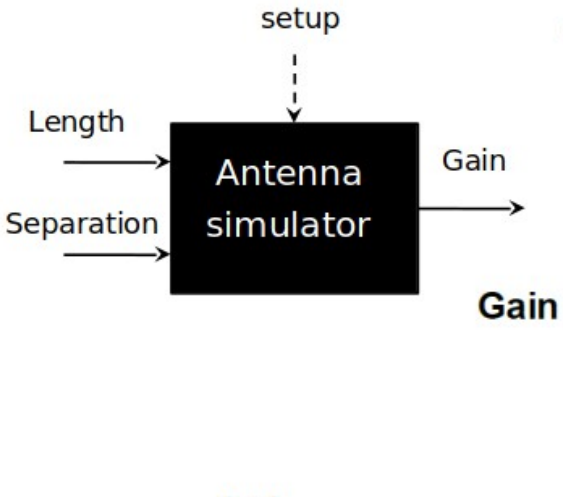
fitness score = 14

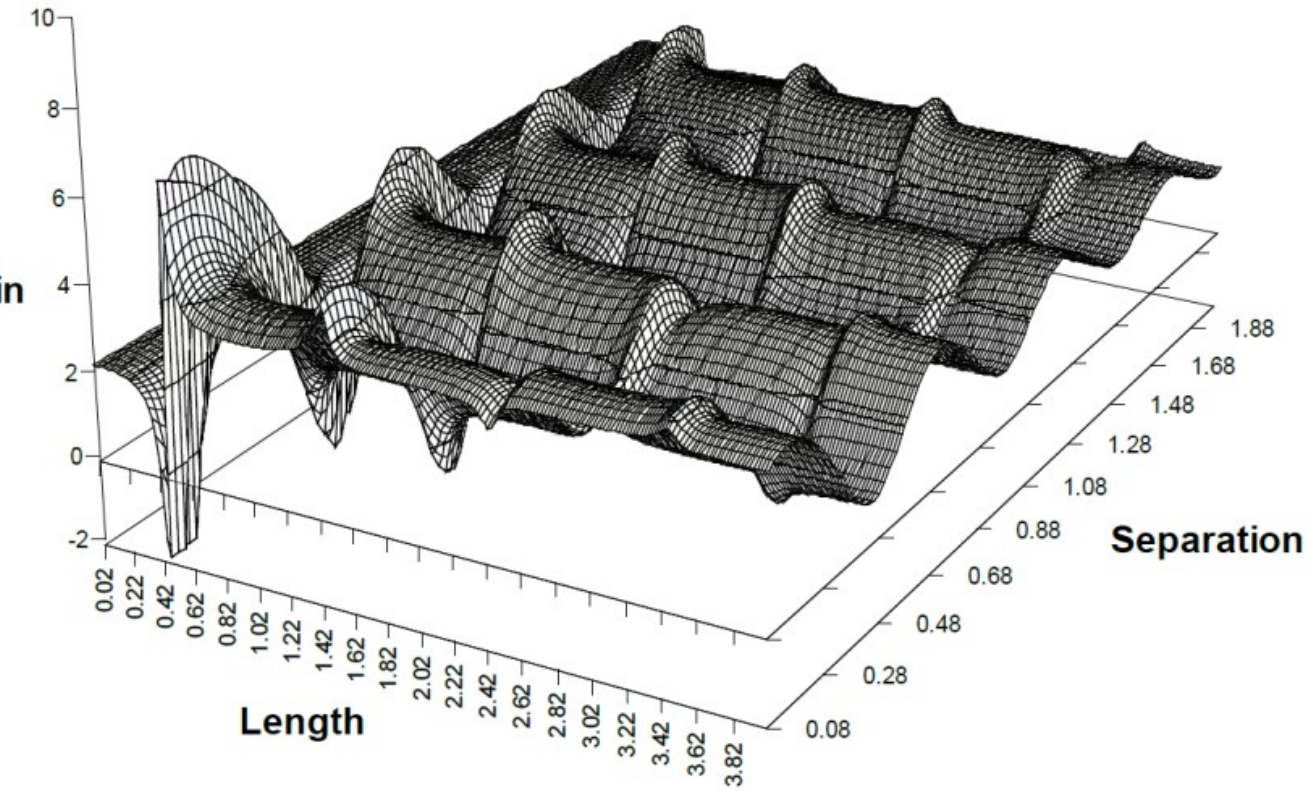
where the fitness function is the number of correct bits with respect to the truth table:

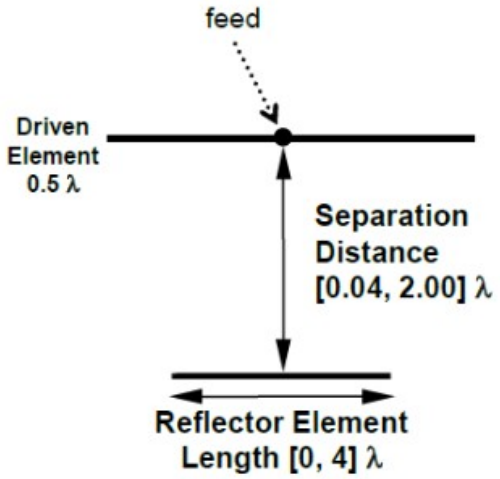
$y(a, b, c, d) = a \vee b \vee c \vee d$.

(max. fitness is 16)

# Example 2: an antenna optimization, a search space illustration




Linden, 1997

What are we looking for?

**Length:** 7 bits
**Separation:** 7 bits

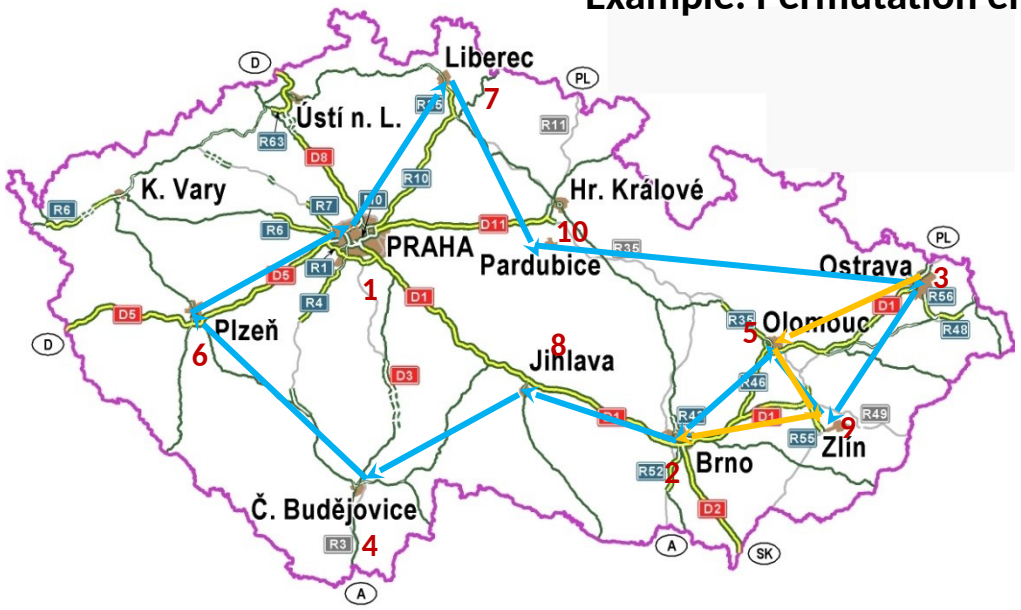
**Search space size**: $2^{14} = 16384$

The **Gain** is our **objective** (**fitness**) **function** – to be maximized.

**In real applications, the search space is usually much larger and multidimensional. It cannot be visualized, its form is unknown!**

# Example 3: encoding a route for Traveling Salesman Problem

**Example: Permutation encoding for TSP**




1 – Praha
2 – Brno
3 – Ostrava
4 – České Budějovice
5 – Olomouc
6 – Plzeň
7 – Liberec
8 – Jihlava
9 – Zlín
10 – Pardubice

Encoded route

| 7 | 10 | 3 | 9 | 5 | 2 | 8 | 4 | 6 | 1 |
|---|---|---|---|---|---|---|---|---|---|

mutation

| 7 | 10 | 3 | 5 | 9 | 2 | 8 | 4 | 6 | 1 |
|---|---|---|---|---|---|---|---|---|---|

**Notice:** genetic operators have to be devised specifically for the given representation (e.g. the mutation for permutations: swap two randomly selected integers; the crossover operator is usually not used here because often the offspring are not valid permutations!).

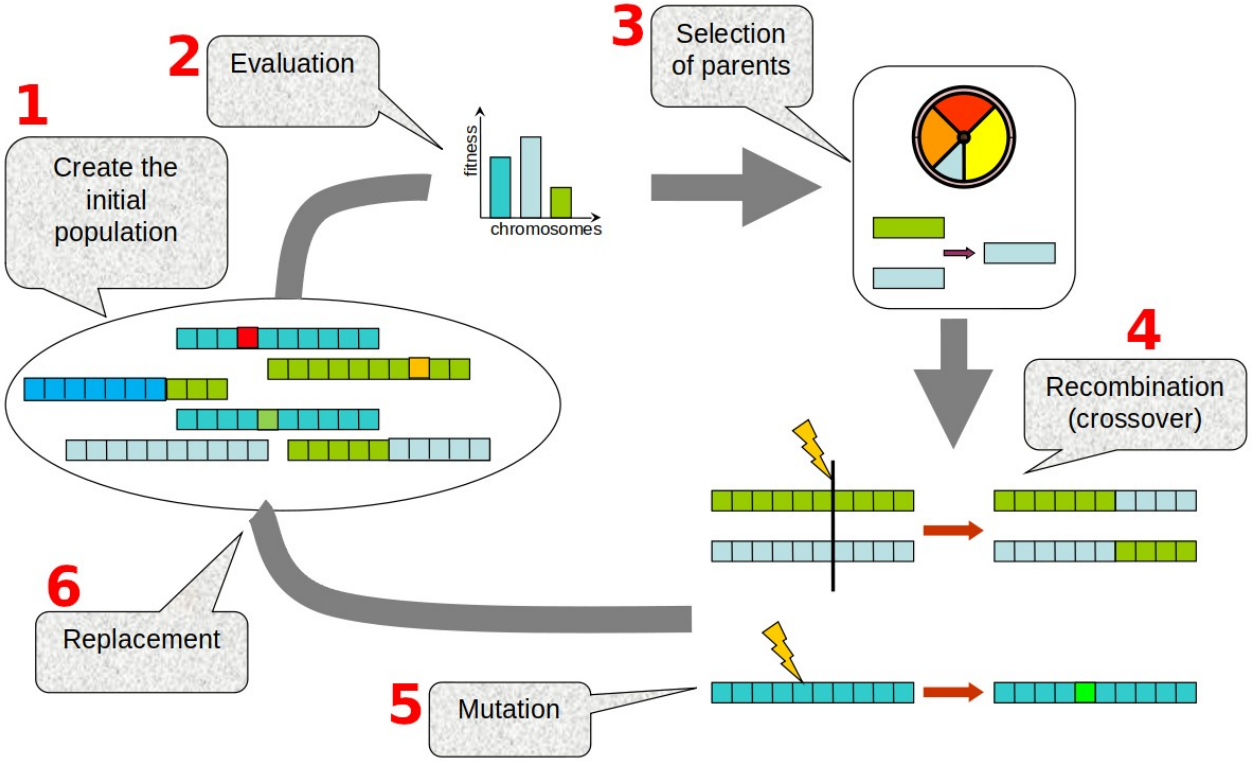
- Examples of the most common encodings (in general):
  - a vector of integers, characters, real numbers… (in computer always have a bin. form)
  - a permutation – see the TSP example above
  - a syntax tree – in GP
  - an acyclic directed graph – in CGP

# The main loop of EA

1. Generate the initial population (randomly or heuristically).
2. Evaluate each candidate solution using the fitness function.

   WHILE termination_condition_is_not_satisfied DO
   - 3. Select parents (according to the fitness).
   - 4-5. Create offspring from parents using crossover and mutation.
   - 6. Create a new population by replacing some/all old individuals by the offspring individuals.

   Evaluate the new population.



# Selection of parents

- Solutions with better fitness have higher probability to be selected.
- Major methods:
  - Weighted roulette wheel selection (fitness proportional) – a relative fitness is calculated for every individual: $f_r(i) = f(i) / \Sigma f(i)$, $i = 1 \ldots$ pop_size

    Individuals are selected proportionally to $f_r$.
    - Problem: If there is an individual with very high $f_r$ and remaining ones have very low $f_r$, the best one will be selected all the time ⇒ degeneration of the population
  - Rank selection – similar to the roulette, but the candidate solutions are sorted according to the fitness first, then a "rank" values are assigned to them based on the ordering (not on the absolute fitness values).
  - Tournament selection – randomly chooses *K* individuals from the population and selects one with the best fitness score as a parent. Often *K* = *2*. Simple, efficient and most widely used.
- The selection procedure is repeated until the required number of parent individuals is obtained.
- An individual can be selected multiple times.
- The selection of the currently best-scored individual is not guaranteed.

# Example 4: selection using the weighted roulette wheel

**A sample populations & fitness:**

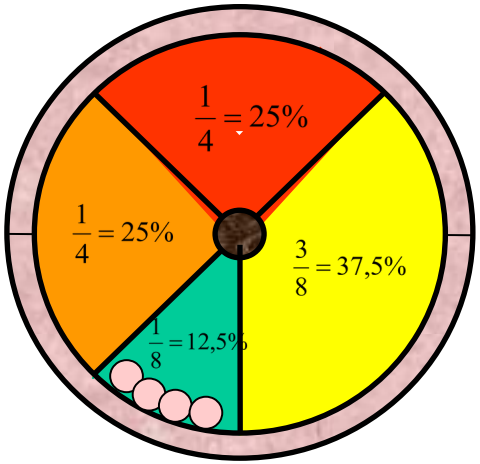
| | | | | |
|---|---|---|---|---|
| 1 | 1 | 0 | 0 | -> fitness (12) = 6 |
| 1 | 0 | 0 | 0 | -> fitness (8) = 4 |
| 0 | 1 | 0 | 1 | -> fitness (5) = 4 |
| 0 | 0 | 1 | 1 | -> fitness (3) = 2 |

The number of portions

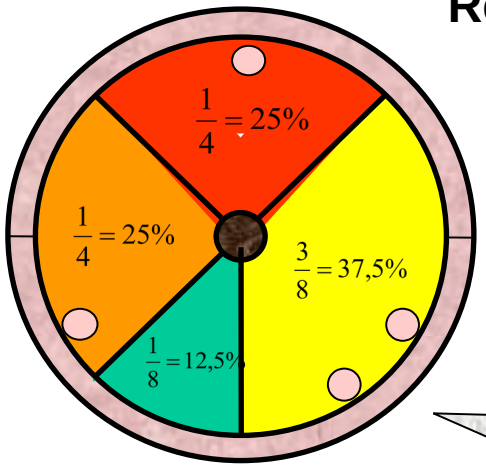
6 + 4 + 4 + 2 = **16**

Calculated portions for the individuals on the wheel

| | | | | |
|---|---|---|---|---|
| 1 | 1 | 0 | 0 | $\rightarrow \frac{6}{16} = \frac{3}{8} = 37{,}5\%$ |
| 1 | 0 | 0 | 0 | $\rightarrow \frac{4}{16} = \frac{1}{4} = 25\%$ |
| 0 | 1 | 0 | 1 | $\rightarrow \frac{4}{16} = \frac{1}{4} = 25\%$ |
| 0 | 0 | 1 | 1 | $\rightarrow \frac{2}{16} = \frac{1}{8} = 12{,}5\%$ |




very unlikely




typical result

**Resulting parents:**

| | | | |
|---|---|---|---|
| 1 | 1 | 0 | 0 |
| 1 | 1 | 0 | 0 |
| 0 | 1 | 0 | 1 |
| 1 | 0 | 0 | 0 |

In order to obtain 4 individuals, the roulette is run 4 times.

# Genetic operators – crossover

Parents | Offspring

**Chromosome 1:** 1 0 0 1 1 | 0 0 1 → 1 0 0 1 1 1 0 1

**Chromosome 2:** 0 1 1 0 1 | 1 0 1 → 0 1 1 0 1 0 0 1

a) Single point

**Chromosome 1:** 1 0 0 | 1 1 | 0 0 1 → 1 0 0 0 1 0 0 1

**Chromosome 2:** 0 1 1 | 0 1 | 1 0 1 → 0 1 1 1 1 1 0 1

b) Double point

**Chromosome 1:** 1 0 0 1 1 0 0 1 → 1 1 1 1 1 0 0 1

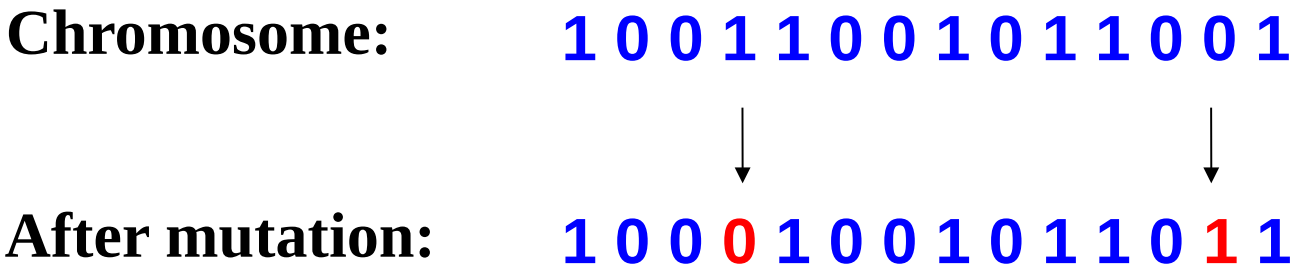
**Chromosome 2:** 0 1 1 0 1 1 0 1 → 0 1 1 1 1 1 0 1

c) Uniform – every bit considered separately

**In EA the crossover is usually applied with a probability, e.g. ~0.7.**

# Genetic operators – mutation

- Mutation is typically a small change in a single chromosome – e.g. a bit inversion:




**In EA the mutation is usually applied with a low probability, e.g. ~0.05. The mutation probability may be higher (sometimes even → 1,0) if the mutation is used as a single genetic operator. However, the mutation always should modify only a very small portion of the chromosome.**

# Replacement and termination criteria

- Replacement mechanisms
  - Steady-state EA: The new population of P individuals will contain N individuals from the old population and M new offspring individuals (P = N + M).
  - Generational EA: The new population contains only new offspring individuals.

- Elitism – the best *e* individuals of the previous population are always copied to the new population (usually *e* = 1).

- Typical termination criteria
  - A solution with fitness = 100% was discovered.
  - The maximum number of generations or time was exhausted.
  - There has not been any progress in the fitness in the last *k* generations.

# Summary of EA utilization

- In order to apply EA to solve a given problem, the designer needs to design:
  - problem encoding
  - genetic operators
  - fitness function

- Then some parameters have to be set up:
  - population size
  - max. number of generations
  - probabilities of mutation, crossover…
  - the way of initialization of the population
  - the termination condition

- As EA are stochastic algorithms, the results need to be evaluated statistically:
  - for each EA settings run the EA several times (at least 20x, better 100x)
  - report the best / average / median / standard deviation of the results

- **Tune the parameters or repeat the entire process if EA does not work.**

In case of real-world applications, there is no relevant theory for solving these issues!
**The design and tuning of EA is primarily an experimental work.**

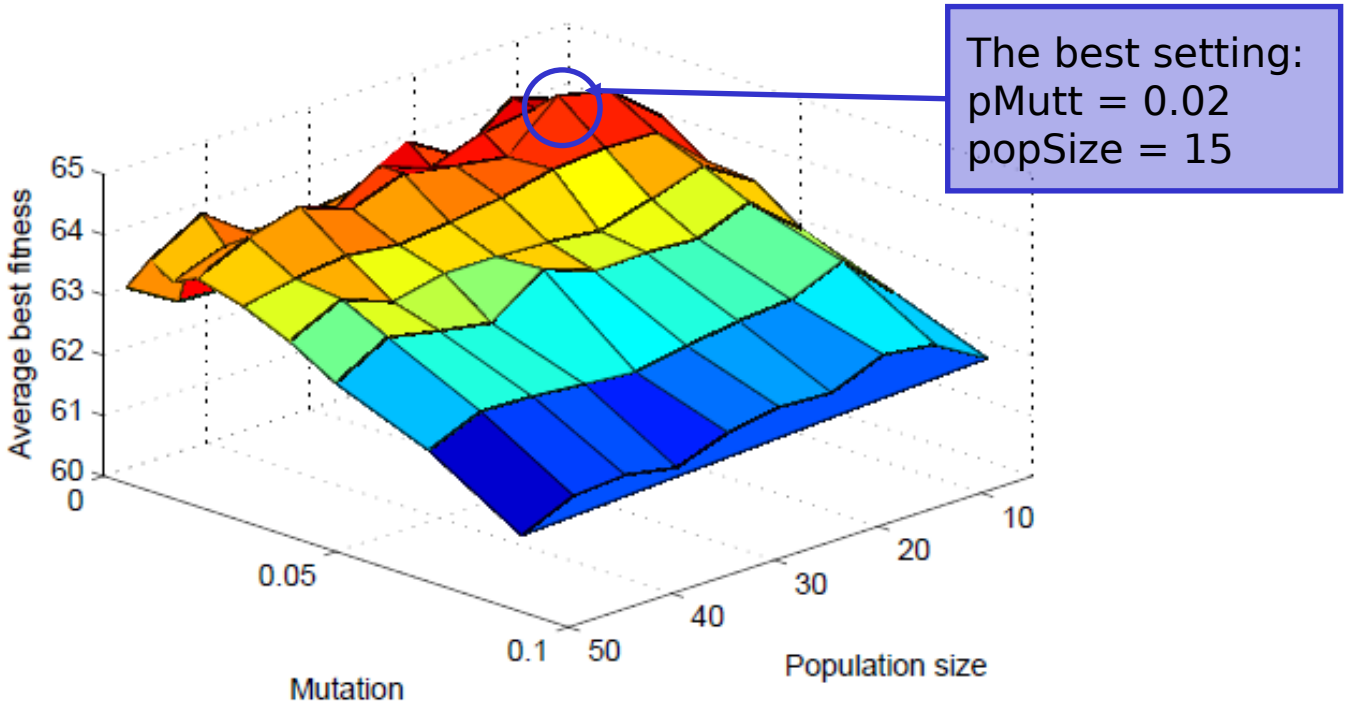
# Is EA1 better than EA2?

- For EA1: P1 is the population size and G1 is the max. number of generations, analogically P2 and G2 for the EA2.
- Then P1*G1 (resp. P2*G2) is the number of candidate solutions evaluated within a single EA run.
- In order to adequately compare EA1 and EA2, **P1*G1 = P2*G2 must hold! Why?** Because if more individuals can be evaluated for an EA (i.e. a bigger part of the search space can be explored), then this EA has a higher chance to provide better results (the comparison would be inadequate).
- **A single run is not meaningful** (may be a lucky chance)!
- The quality of EA can be statistically measured as (for *r* independent runs):
  - the average (median) number of evaluations (or generations) needed to find an acceptable solution (given by the termination condition),
  - the number of successful runs (when 100% fitness was achieved) out of *r* runs,
  - the average (median) fitness of the best solutions after the max. generations,
  - and many other methods exists.

# Example 5: visualization of statistical tuning of some EA parameters

Principles of the statistical evaluation - an example:

For 10 different values of P, the corresponding values G=40000/P, and 10 values of the mutation probability, we performed 50 independent runs **for each (P, G, Pmut)**. All other EA parameters remain fixed. In total, 50 x 10 x 10 independent runs are performed. The average best fitness out of 50 independent runs for each settings is visualized.



# Outline

- Introduction
- Evolutionary algorithms
- Genetic programming
- Cartesian genetic programming
- Applications from FIT
- Summary

# Genetic Programming (GP)

- Genetic Programming is an EA for the evolutionary design of computer programs.
- The goal is to find a program (close to) optimally working for arbitrary input data and providing the desired results.
- In GP:
  - **The EA is almost identical with that on page 15.**
  - Ccandidate programs are represented using syntax trees (or other suitable structures), machine code instructions or strings of integers… all representing an executable (calculable) function.
  - Crossover and mutation are adapted adequately, sometimes more complex genetic operators can be used, e.g. subprogram definition.
  - In order to obtain the fitness score, a candidate program is executed using some given training data.
  - The result of GP (i.e., the best-performing program) is evaluated using the test data.

# GP – preparatory steps

- In addition to that stated on page 21, we need to define:

  A set of terminals (T), typically: constants, input variables, functions of 0-arity (e.g. time(), rand(), halt() etc.) - the terminals will appear only in the leafs of the trees.

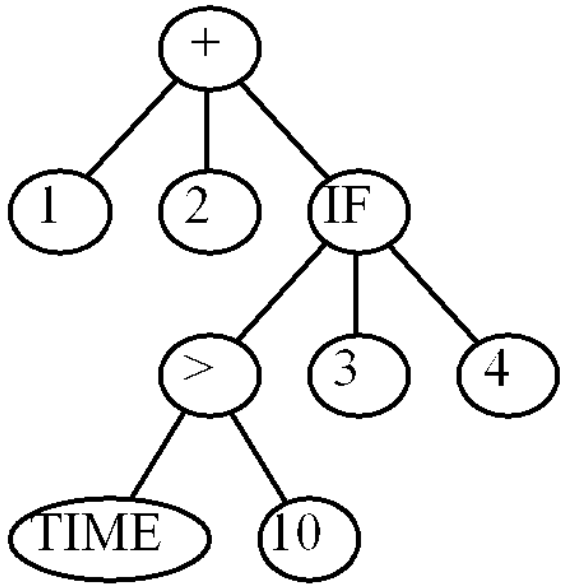
  A set of functions (F) - non-terminals, typically: +, -, *, , *AND*, *NOT*, … *any function with at least 1 input – these functions will appear in non-leaf nodes of the trees.*

  ***Note**: some operations need to be defined as **protected functions** in order to avoid program crash during execution, e.g. division:*

  *div(a, b) := if b != 0 then return a/*b else return 1

# Example 6: a sample GP tree

- Consider T = { 0,..., 9, TIME }, F = { +, -, >, <, IF }
- A syntax tree may look like this, e.g.:
- In GP, however, the trees are represented in a suitable language, e.g. LISP:




```
(+ 1 2 (IF (> TIME 10) 3 4))
```

- Or the same may be written in C:

```
int a;
if (TIME() > 10) a = 3 else a = 4;
return 1 + 2 + a;
```

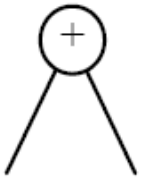
# GP – random program generation

- Given a maximum tree depth = h, the generation starts from the root node.
- Methods for generating the initial population of trees:
  - Grow – randomly choose items from T or F until h is reached, then use only T.
  - Full – randomly choose items only from F until h is reached, then use only T.
  - Ramped Half-and-Half – for each depth 1…h generate one half of individuals in the population by the Grow method and the other half of individuals by the Full method.

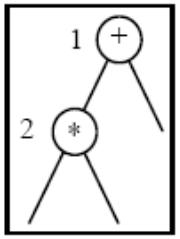
# Example 7: a random tree generation

- Terminal set T = {A, B, C}
- Function set F = {+, –, *, %, IFLTE}

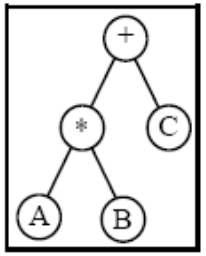
**BEGIN WITH TWO-ARGUMENT +**




**CONTINUE WITH TWO-ARGUMENT ***



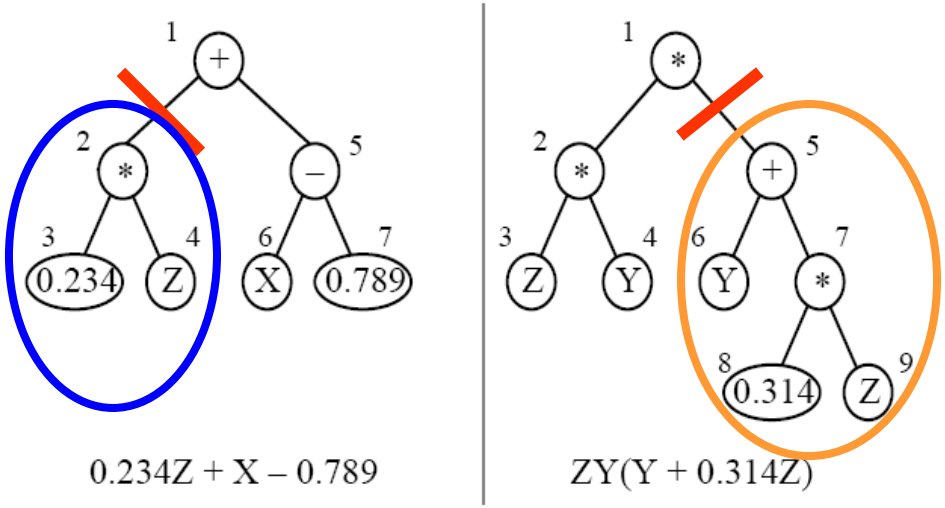

**FINISH WITH TERMINALS A, B, AND C**




- The result is a syntactically valid executable program (provided the set of functions is closed)

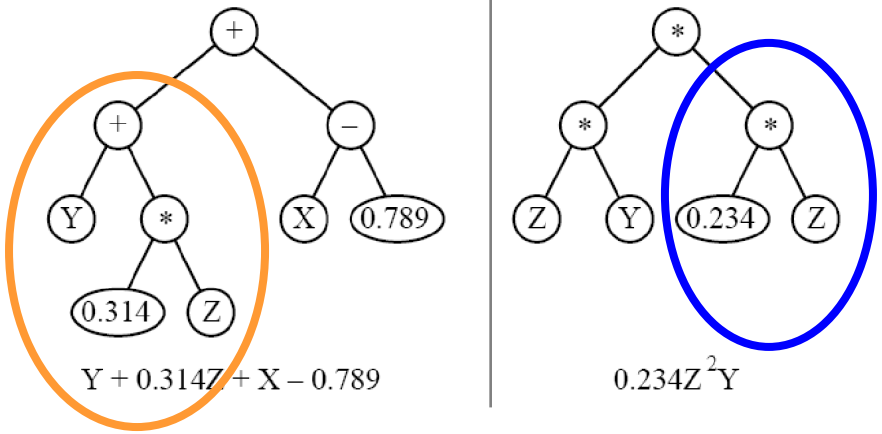
# GP: Crossover

## 2 parents



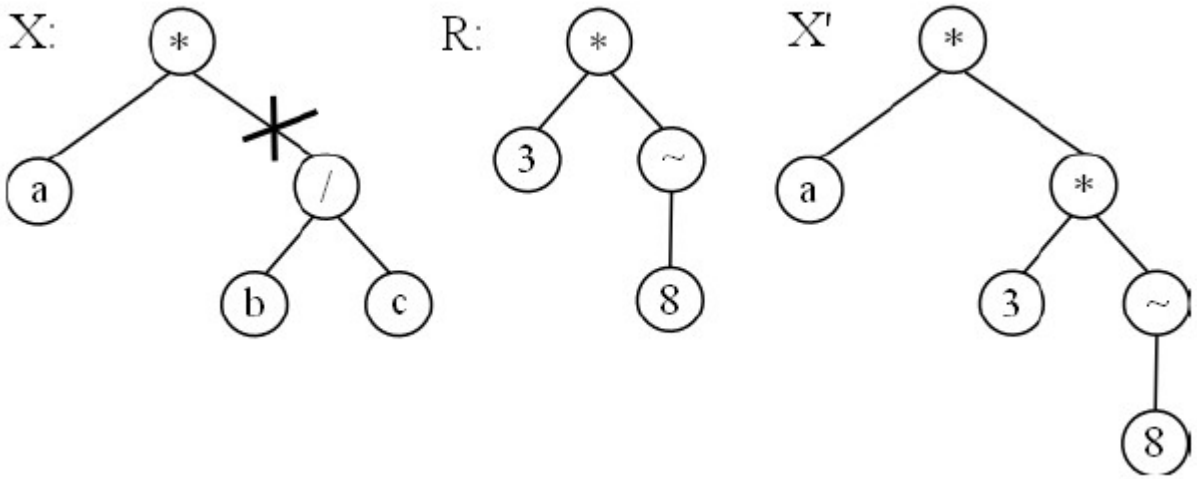

Crossover swaps randomly selected sub-trees.

## 2 offspring



# GP: Mutation

- Randomly select a node (a corresponding sub-tree is marked) in X and replace the sub-tree by a randomly generated sub-tree R to get X'.



# A typical GP application: Symbolic Regression (~ data fitting)

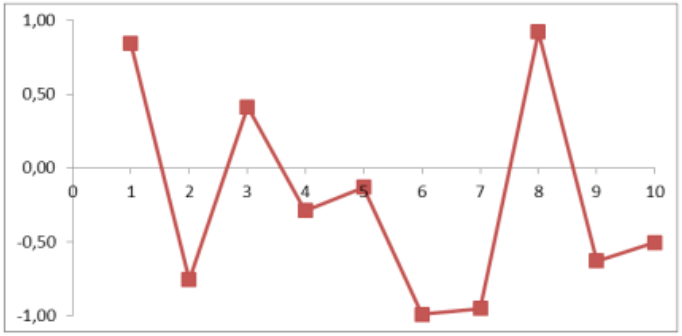
Given some data (usually from a measurement), the goal of symbolic regression is to discover a mathematical prescription that fits the data as best as possible. For example, for 10 test cases given by the table, we are trying to minimize the function *f* (the sum of absolute differences):

$$f = \sum_{i=1}^{K=10} |y_i - w_i|$$

$K=10$ ← The number of test cases

$y_i$: Desired result

$w_i$: Candidate program result




Test cases

| x | y |
|---|---|
| 1 | 0,84 |
| 2 | -0,76 |
| 3 | 0,41 |
| 4 | -0,29 |
| 5 | -0,13 |
| 6 | -0,99 |
| 7 | -0,95 |
| 8 | 0,92 |
| 9 | -0,63 |
| 10 | -0,51 |

Why to use GP?

- An arbitrary function set can be used (no assumptions in terms of mathematical properties).
- Can be extended to n-input/m-output functions and many objectives can be optimized together (error, size of the tree, speed...).
- Constraints can easily be handled.

Disadvantages: computationally expensive, stochastic

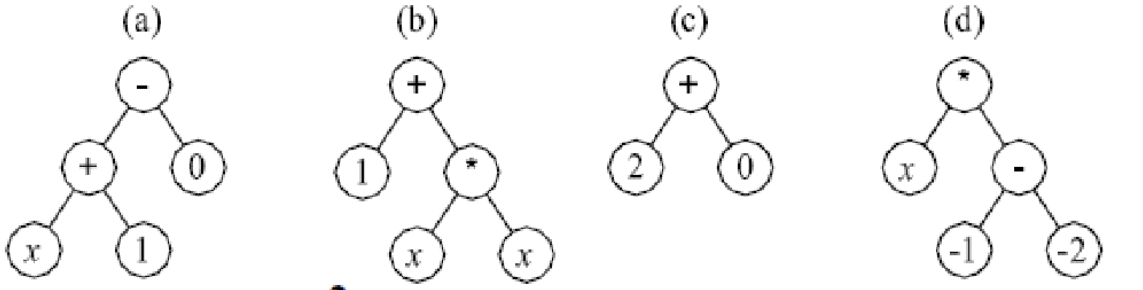
# Example 8: a simple example of symbolic regression using GP

| Independent variable $X$ | Dependent variable $Y$ |
|---|---|
| -1.00 | 1.00 |
| -0.80 | 0.84 |
| -0.60 | 0.76 |
| -0.40 | 0.76 |
| -0.20 | 0.84 |
| 0.00 | 1.00 |
| 0.20 | 1.24 |
| 0.40 | 1.56 |
| 0.60 | 1.96 |
| 0.80 | 2.44 |
| 1.00 | 3.00 |

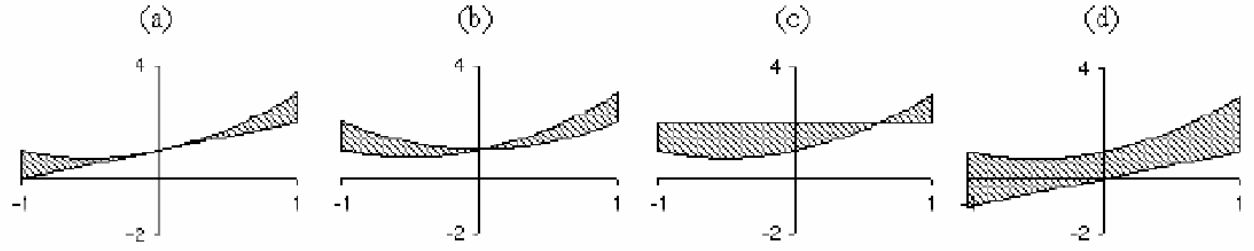
| | | |
|---|---|---|
| | **Objective:** | **Find a computer program with one input (independent variable $x$), whose output equals the value of the quadratic polynomial $x^2 + x + 1$ in range from -1 to +1.** |
| 1 | **Terminal set:** | `T = {X}` |
| 2 | **Function set:** | `F = {+, -, *, %}`<br>**NOTE: The protected division function % returns a value of 1 when division by 0 is attempted (including 0 divided by 0)** |
| 3 | **Fitness:** | **The sum of the absolute value of the differences (errors), computed (in some way) over values of the independent variable $x$ from –1.0 to +1.0, between the program's output and the target quadratic polynomial $x^2 + x + 1$.** |
| 4 | **Parameters:** | **Population size $M = 4$.** |
| 5 | **Termination:** | **An individual emerges whose sum of absolute errors is less than 0.1** |

# Example 8: a simple example of symbolic regression using GP

**GENERATION 0**

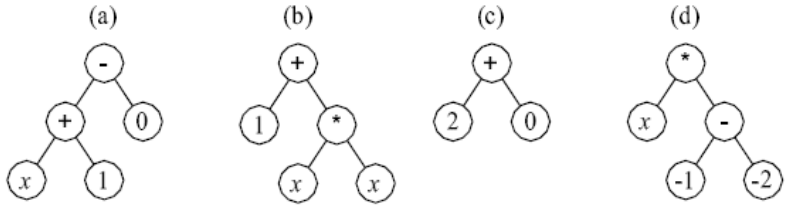



**FITNESS**

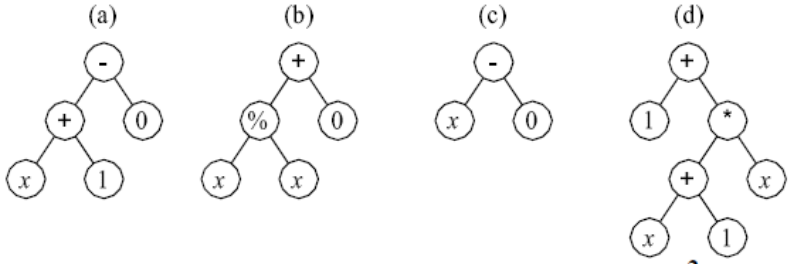


# Example 8: a simple example of symbolic regression using GP

**GENERATION 0**




## GENERATION 1




| $x + 1$ | 1 | $X$ | $x^2 + x + 1$ |
|---|---|---|---|
| Copy of (a) | Mutant of (c) —picking "2" as mutation point | First offspring of crossover of (a) and (b) —picking "+" of parent (a) and left-most "x" of parent (b) as crossover points | Second offspring of crossover of (a) and (b) —picking "+" of parent (a) and left-most "x" of parent (b) as crossover points |

# Example 9: a more complex example of symbolic regression…

**SYMBOLIC REGRESSION OF QUARTIC POLYNOMIAL $X^4+X^3+X^2+X$ (WITH 21 FITNESS CASES)**

| Independent variable $X$ (Input) | Dependent Variable $Y$ (Output) |
|---|---|
| -1.0 | 0.0000 |
| -0.9 | -0.1629 |
| -0.8 | -0.2624 |
| -0.7 | -0.3129 |
| -0.6 | -0.3264 |
| -0.5 | -0.3125 |
| -0.4 | -0.2784 |
| -0.3 | -0.2289 |
| -0.2 | -0.1664 |
| -0.1 | -0.0909 |
| 0 | 0.0 |
| 0.1 | 0.1111 |
| 0.2 | 0.2496 |
| 0.3 | 0.4251 |
| 0.4 | 0.6496 |
| 0.5 | 0.9375 |
| 0.6 | 1.3056 |
| 0.7 | 1.7731 |
| 0.8 | 2.3616 |
| 0.9 | 3.0951 |
| 1.0 | 4.0000 |

**TABLEAU—SYMBOLIC REGRESSION OF QUARTIC POLYNOMIAL $X^4+X^3+X^2+X$**

| | |
|---|---|
| Objective: | Find a function of one independent variable, in symbolic form, that fits a given sample of 21 $(x_i, y_i)$ data points |
| Terminal set: | x (the independent variable). |
| Function set: | `+, -, *, %, SIN, COS, EXP, RLOG` |
| Fitness cases: | The given sample of 21 data points $(x_i, y_i)$ where the $x_i$ are in interval [–1,+1]. |
| Raw fitness: | The sum, taken over the 21 fitness cases, of the absolute value of difference between value of the dependent variable produced by the individual program and the target value $y_i$ of the dependent variable. |
| Standardized fitness: | Equals raw fitness. |
| Hits: | Number of fitness cases (0 – 21) for which the value of the dependent variable produced by the individual program comes within 0.01 of the target value $y_i$ of the dependent variable. |
| Wrapper: | None. |
| Parameters: | Population size, $M = 500$.<br>Maximum number of generations to be run, $G = 51$. |
| Success Predicate: | An individual program scores 21 hits. |

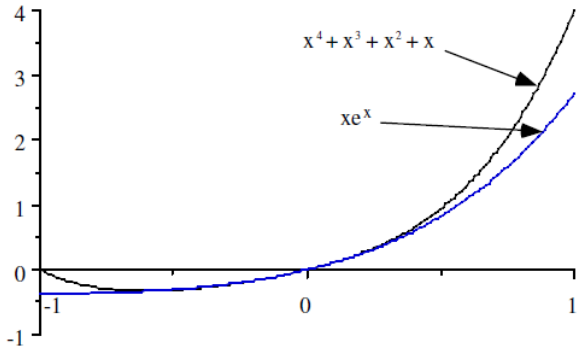
# Example 9: a more complex example of symbolic regression…

**BEST-OF-GENERATION INDIVIDUAL IN GENERATION 0 WITH RAW FITNESS OF 4.47 (AVERGAGE ERROR OF 0.2)**

```
(* X (+ (+ (- (% X X) (% X X))
(SIN (- X X))) (RLOG (EXP (EXP
X)))))
```

**Equivalent to**

$$xe^x$$




**CREATION OF GENERATION 1 FROM GENERATION 0**

- In the so-called "generational" model for genetic algorithms, a new population is created that is equal in size to the old population
  - 1% mutation (i.e., 5 individuals out of 500)
  - 9% reproduction (i.e., 45 individuals)
  - 90% crossover (i.e., 225 pairs of parents — yielding 450 offspring)

- All participants in mutation, reproduction, and crossover are chosen from the current population <u>PROBABILISTICALLY, BASED ON FITNESS</u>
  - Anything can happen
  - Nothing is guaranteed
  - The search is heavily (but not completely) biased toward high-fitness individuals
  - The best is not guaranteed to be chosen
  - The worst is not necessarily excluded
  - Some (but not much) attention is given even to low-fitness individuals

# Example 9: a more complex example of symbolic regression…

**BEST-OF-GENERATION INDIVIDUAL IN GENERATION 2 WITH RAW FITNESS OF 2.57 (AVERGAGE ERROR OF 0.1)**

```
(+ (* (* (+ X (* X (* X (% (% X
X) (+ X X)))))
          (+ X (* X X))) X) X)
```

**Equivalent to...**

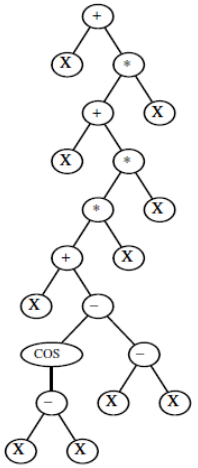
$$x^4 + 1.5x^3 + 0.5x^2 + x$$

**BEST-OF-RUN INDIVIDUAL IN GENERATION 34 WITH RAW FITNESS OF 0.00 (100%-CORRECT)**

```
(+ X (* (+ X (* (* (+ X (- (COS
(- X X)) (- X X))) X) X)) X))
```

**Equivalent to**

$$x^4 + x^3 + x^2 + x$$




**Notice:** COS (X – X), (X – X) etc. are examples of the so-called **intron** – a piece of code which does not contribute to the fitness, but increases the size of candidate solution. COS (X – X) = 1, (X – X) = 0 for any X etc.

**Bloat** – a phenomenon in which the program size grows during the evolution, but the fitness remains unchanged. Various techniques were developed to eliminate the bloat.

# Outline

- Introduction
- Evolutionary algorithms
- Genetic programming
- Cartesian genetic programming
- Applications from FIT
- Summary

# Cartesian Genetic Programming – CGP [Miller, 1999]

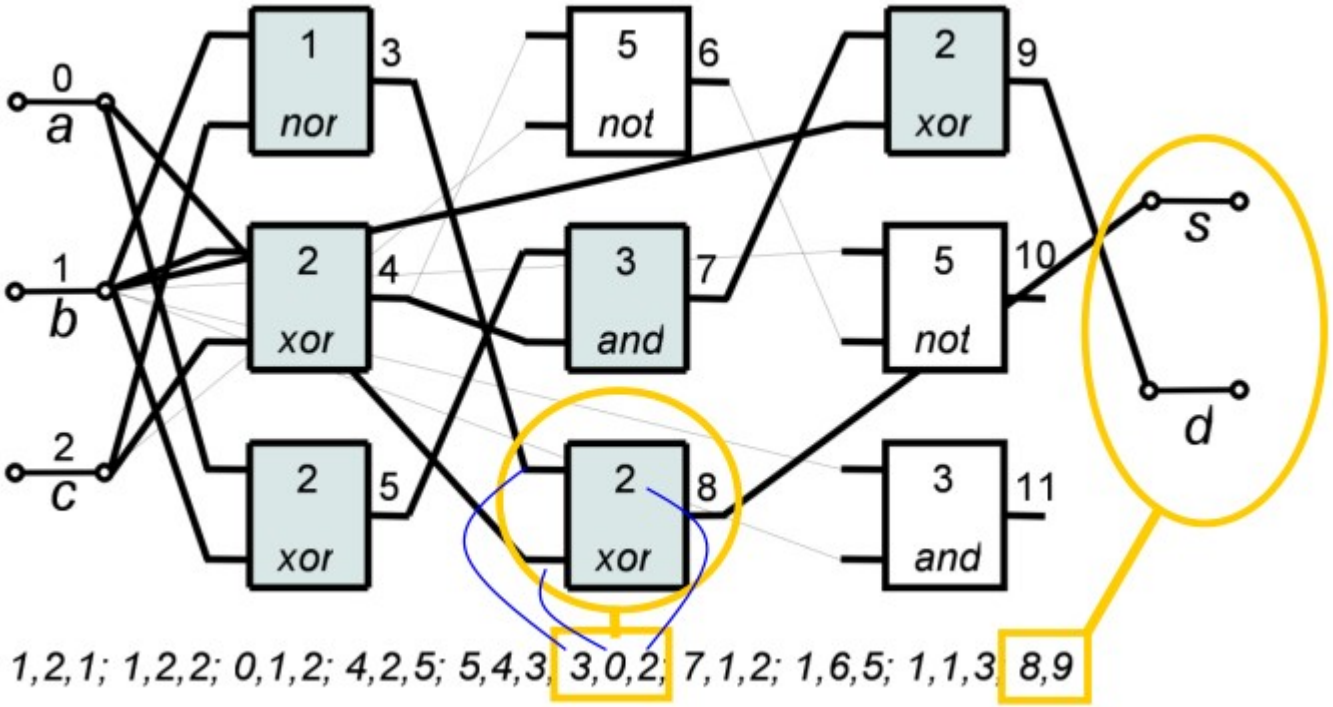
- So far we discussed so-called “tree-based” GP.
- There are also other types (Linear GP, developmental GP…).
- CGP is a type of GP in which the programs are represented as *oriented acyclic graphs* the nodes of which form a matrix.
- CGP encodes these graphs as **fixed length integer vectors**. This representation in particularly suitable for designing digital circuits.
- CGP parameters include:
  - The number of rows, $n_r$
  - The number of columns, $n_c$
  - The number of primary inputs, $n_i$
  - The number of primary outputs, $n_o$
  - The maximum arity of a block function, $n_a$
  - Level-back parameter (L-back), L
  - The set of block functions, Γ

In genotypes, there are $n_a+1$ integers per node (describing connection of node inputs and node function) and $n_o$ integers for describing connection of primary outputs. Therefore, the genomes are of the length $n_c n_r(n_a + 1) + n_o$ integers.

# Example 10: a logic circuit represented in CGP

- Consider $n_r=3$, $n_c=3$, $n_i=3$, $n_o=2$, $n_a=2$, $L=3$,

  $\Gamma = \{NAND^{(0)}, NOR^{(1)}, XOR^{(2)}, AND^{(3)}, OR^{(4)}, NOT^{(5)}\}$

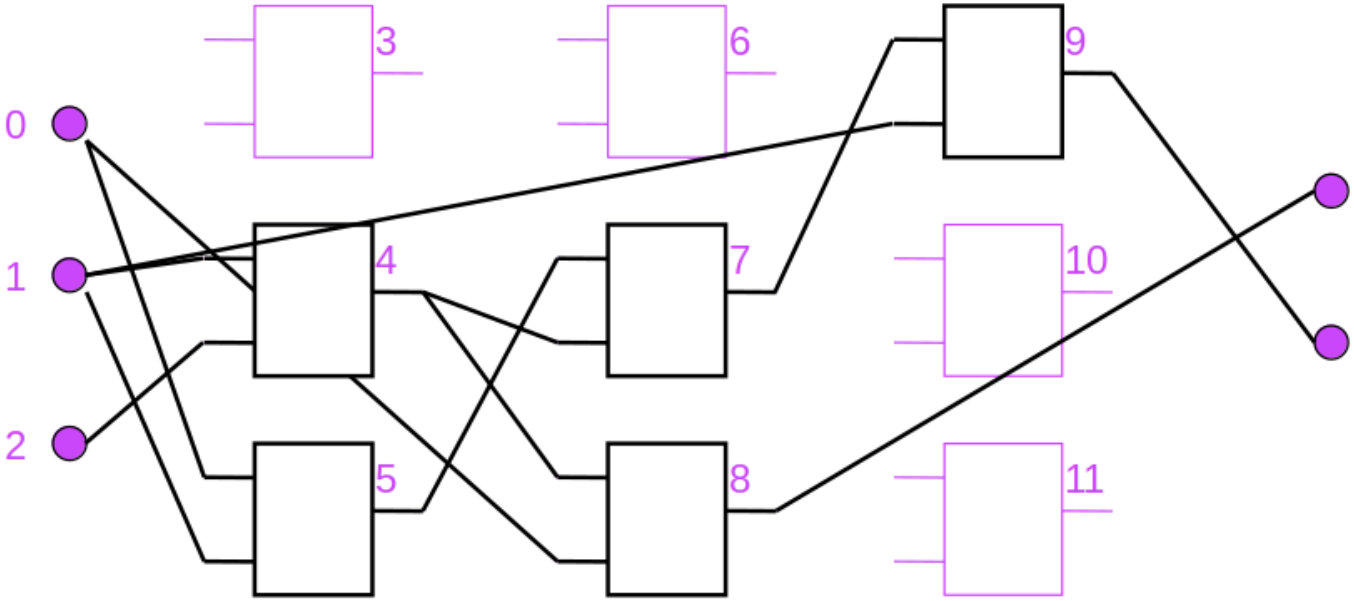


# Circuit construction from a CGP chromosome

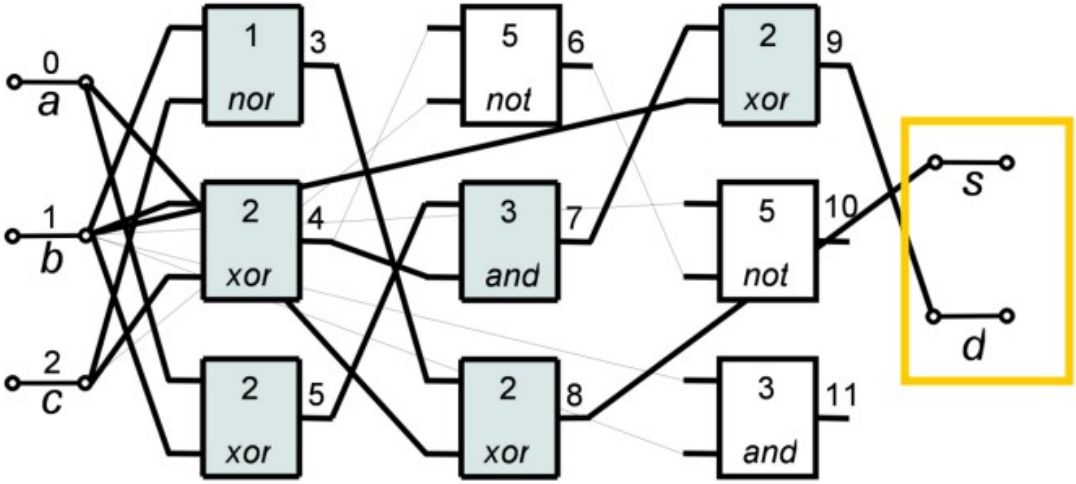
(1) The processing of the chromosome goes from the outputs to the inputs. The nodes that need to be evaluated are stored in a stack.

(2) Then, the functions are taken from the stack and evaluated, the results are stored in an array at the indexes given by the figure.

A chromosome (from the previous example):
(1,2,1) (1,2,2) (0,1,2) (4,2,5) (5,4,3) (4,0,2) (7,1,2) (1,6,5) (1,1,3) (8,9)



# Fitness function for circuit design using CGP




*1,2,1; 1,2,2; 0,1,2; 4,2,5; 5,4,3; 3,0,2; 7,1,2; 1,6,5; 1,1,3; 8,9*

| a | b | c | d | s | |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | |
| 0 | 0 | 1 | 0 | 1 | |
| 0 | 1 | 0 | 0 | 1 | |
| 0 | 1 | 1 | 1 | 0 | |
| 1 | 0 | 0 | 0 | 1 | |
| 1 | 0 | 1 | 1 | 0 | |
| 1 | 1 | 0 | 1 | 0 | |
| 1 | 1 | 1 | 1 | 1 | => fitness = 16 |

Specification (1-bit adder), target table:

| a | b | c | d | s | |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 | |
| 0 | 0 | 1 | 0 | 0 | |
| 0 | 1 | 0 | 0 | 0 | |
| 0 | 1 | 1 | 1 | 0 | |
| 1 | 0 | 0 | 0 | 0 | |
| 1 | 0 | 1 | 1 | 1 | |
| 1 | 1 | 0 | 1 | 1 | |
| 1 | 1 | 1 | 1 | 1 | => fitness = 10 |

## Typical fitness function (to be maximized):

- $$f = Max - \sum_{i=1}^{K} HD(y_i, w_i)$$

  $K$ ← The number of test vectors

  $HD(y_i, w_i)$ ↑ Hamming distance (between circuit response and desired response)

## Additional objectives:

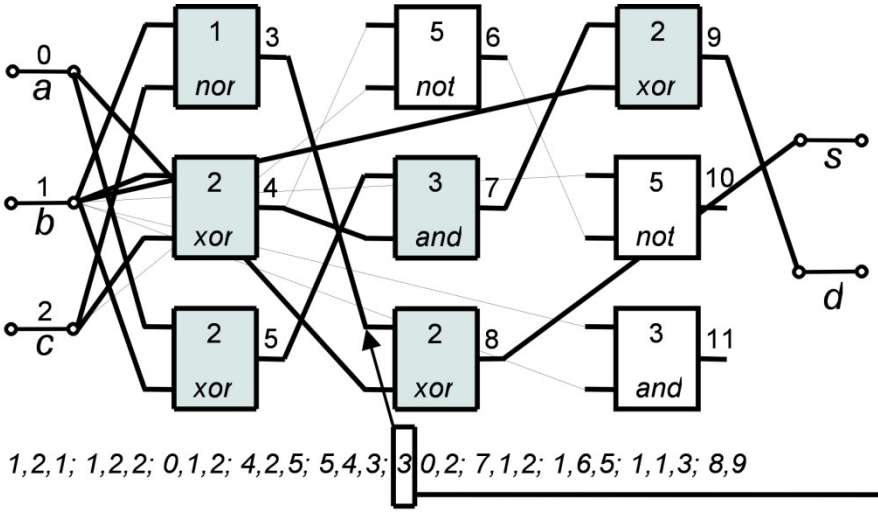
- area (the number of gates)
- delay
- power consumption etc.

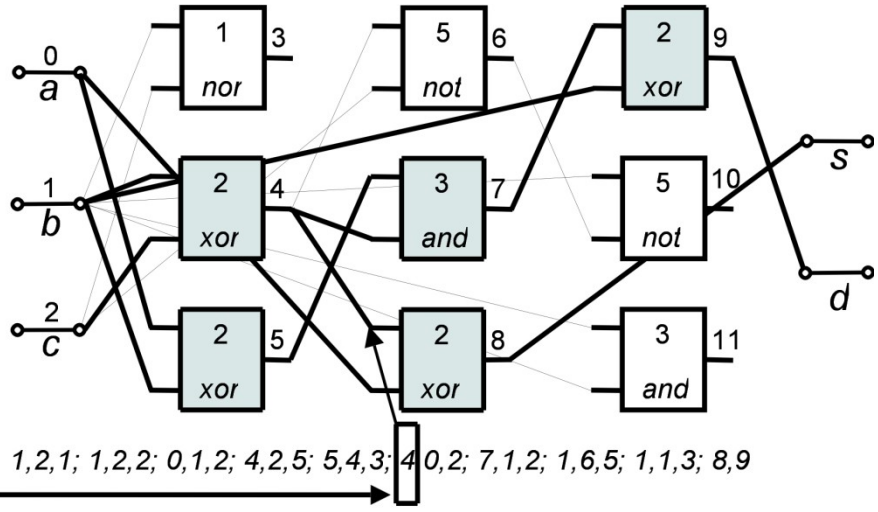
*Max = #outputs * $2^{\#inputs}$ (in our case $2 * 2^8 = 16$)*

*K = $2^{\#inputs}$* for combinational circuits. Not scalable!!!

# CGP uses mutation-based search (usually without crossover)

- Mutation: Randomly select *h* integers and replace them by randomly generated (but legal) values.
- Mutation can be useful, harmful or neutral.




**mutation**

(a)

| a | b | c | d | s |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 | 1 |
| 1 | 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 1 | 1 |

=> fitness = 10

(b)

| a | b | c | d | s |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 1 |
| 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 1 | 1 |

=> fitness = 16
(for full adder)

More on mutations in CGP -> Goldman B.W., Punch W.F.: Analysis of Cartesian Genetic Programming's Evolutionary Mechanisms. IEEE Trans. Evolutionary Computation 19(3): 359-373 (2015)

# EA for CGP: (1 + λ)-Evolution Strategy (see [Miller, 1999])

**Algorithm 1: CGP**

**Input**: CGP parameters, fitness function
**Output**: The highest scored individual $p$ and its fitness

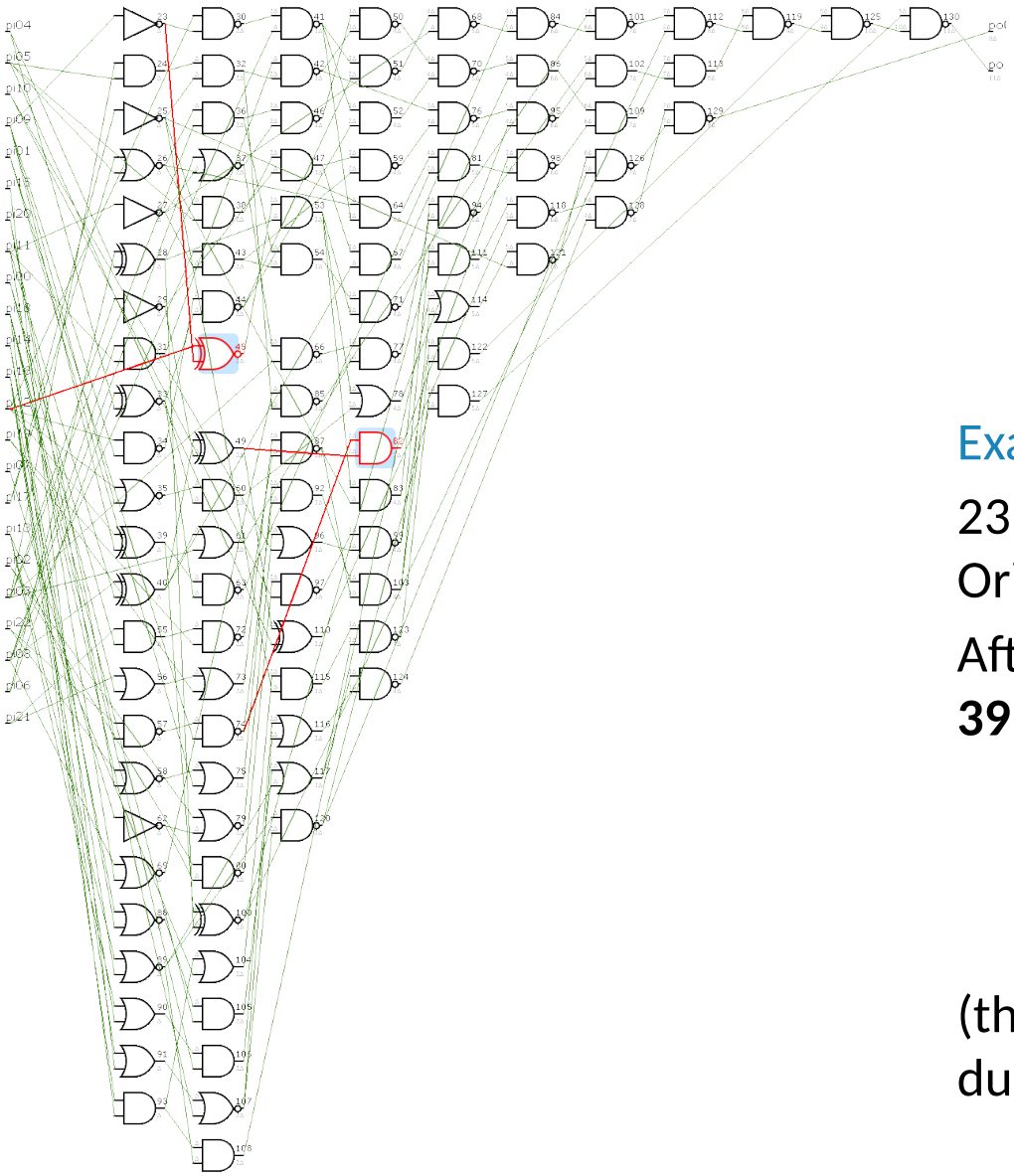
```
1 P ← randomly generate population; // or use conventional designs
2 EvaluatePopulation(P); p ← highest-scored-individual(P);
3 while ⟨terminating condition not satisfied⟩ do
4     α ← highest-scored-individual(P);
5     if fitness(α) ≥ fitness(p) then
6         p ← α;
7     P ← create λ offspring of p using mutation;
8     EvaluatePopulation(P);
9 return p, fitness(p);
```

≥ is better than > as it increases the diversity of the population and leads to better solutions.

# Outline

- Introduction
- Evolutionary algorithms
- Genetic programming
- Cartesian genetic programming
- Applications from FIT
- Summary

# Optimization of the number of gates of existing circuits using CGP



Example: Cordic circuit

23 inputs, 2 outputs

Original: **106 gates** (LGSynth93)

After the CGP optimization:

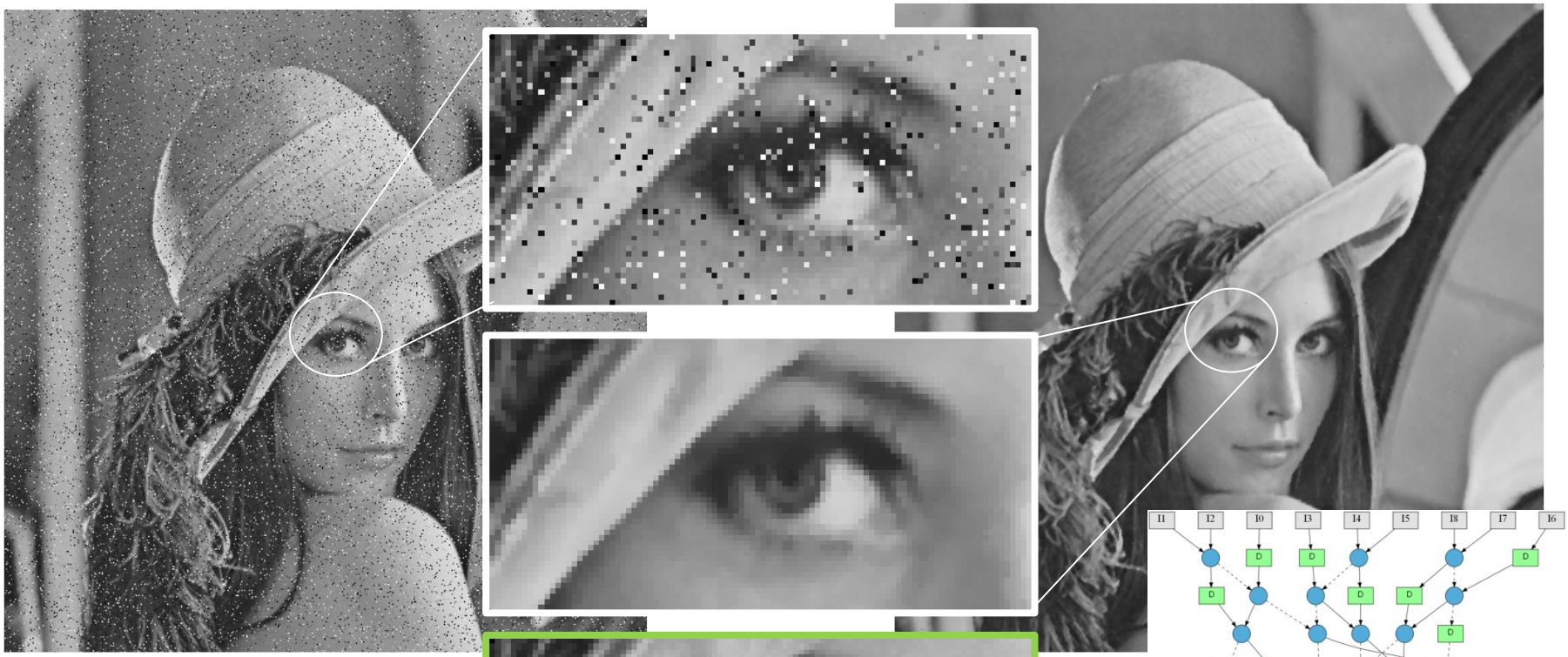
**39 gates**

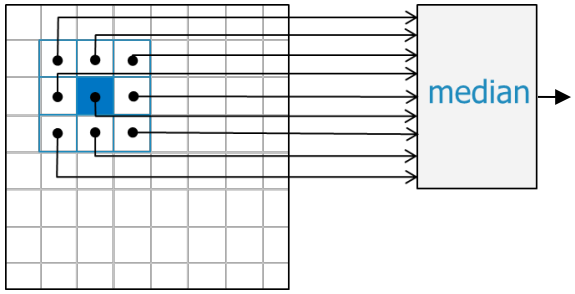
(the best circuits discovered during a CGP run are shown)

# Non-linear image filters

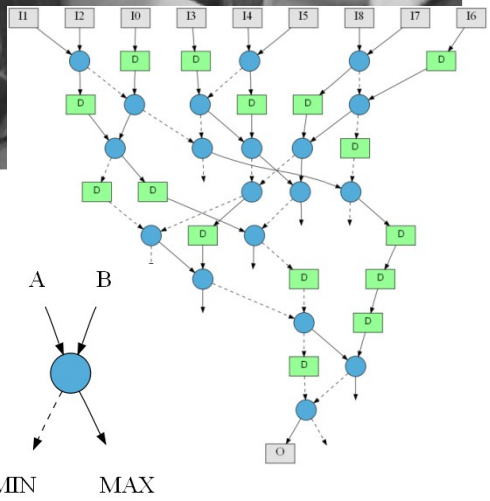
corrupted image
(10% pixels, impulse noise)

filtered image
(9-input median filter)

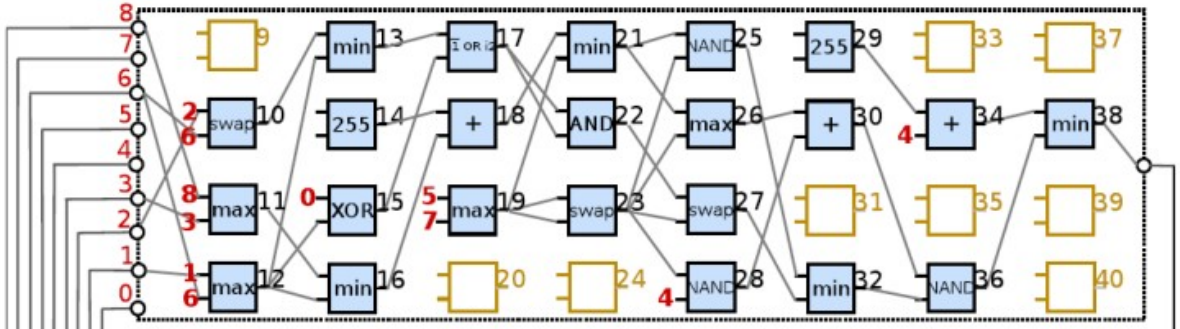





origin



# Evolutionary design of image filters using CGP




Input image

N x M pixels

Output image - *v*

Reference img - *w*

$$fitness = \sum_{i=1}^{N} \sum_{j=1}^{M} |v(i,j) - w(i,j)|$$

- $n_i$=9, n_o=1, $n_c$=8, $n_r$=4, $n_a$=2, λ = 7; 30,000 generations/run; mutation: max. 5%
- Function set (over 8 bits) = {+, min, max, swap, constants, average, log. functions}

# CGP: Evolutionary design with 8-bit components (FPGA)

a) Image corrupted by 5% salt-and-pepper noise

PSNR: 18.43 dB (peak signal to noise ratio)

b) Original image

c) Median filter (kernel 3x3)

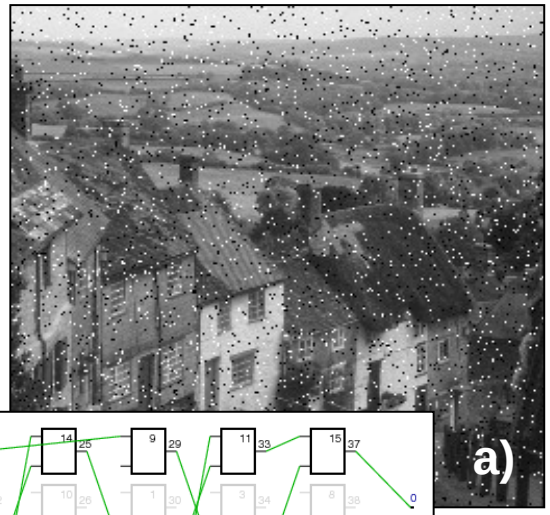
PSNR: 27.92 dB

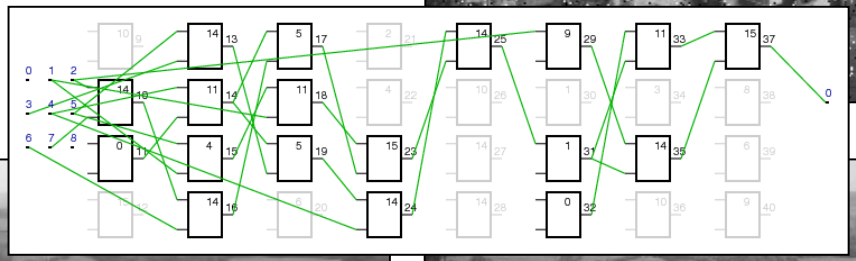
268 FPGA slices; 305 MHz

d) Evolved filter (kernel 3x3)

PSNR: 37.50 dB

200 FPGA slices; 308 MHz



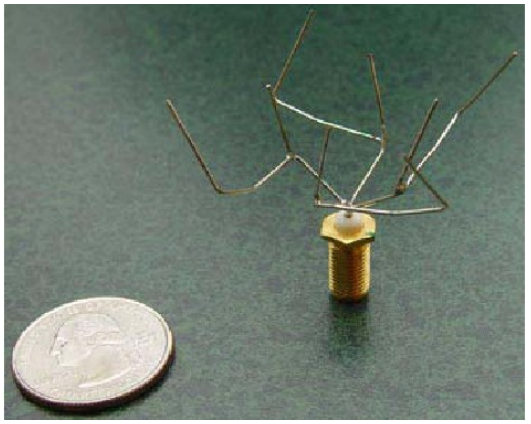





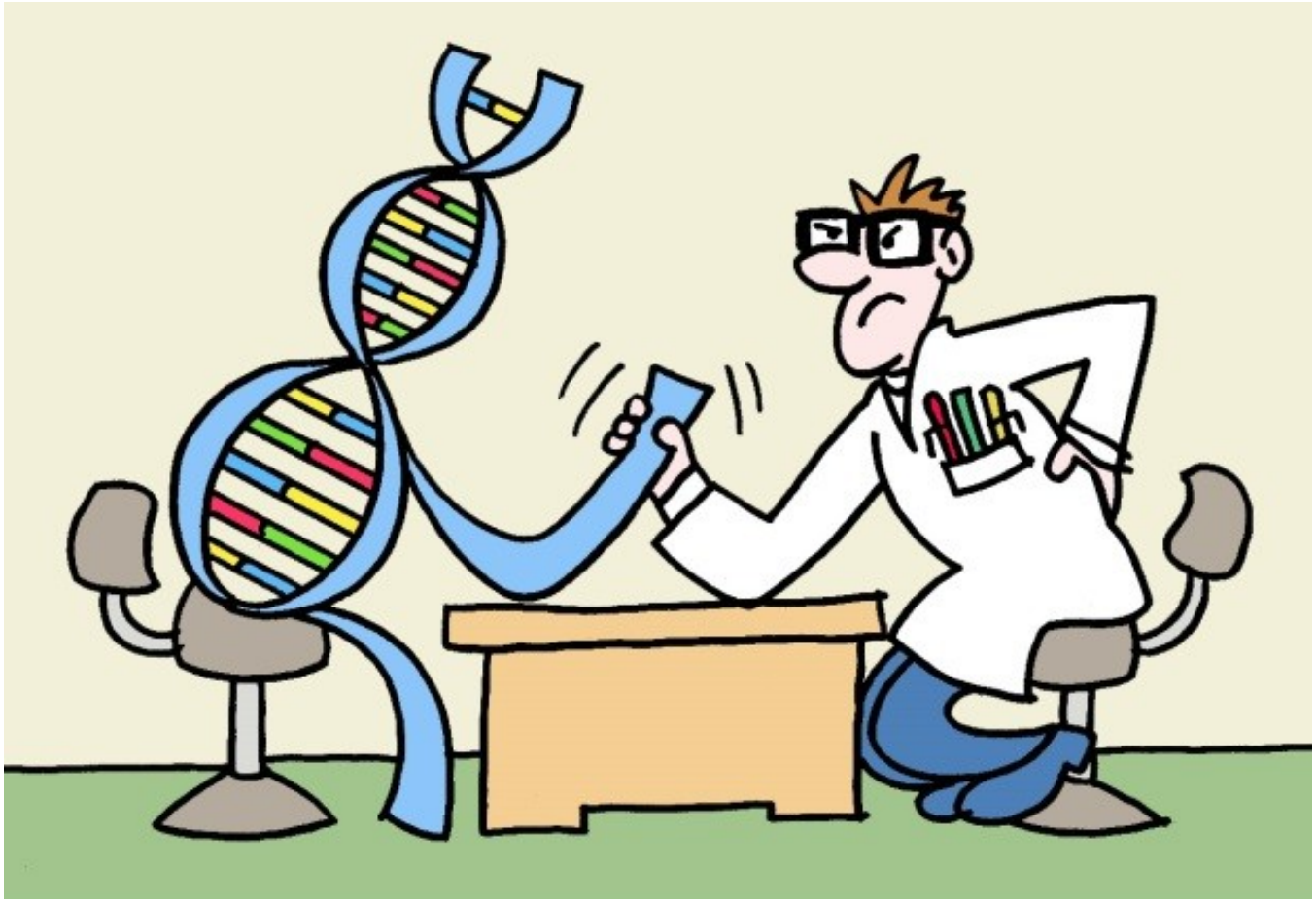

A Czech patent #304181 approved for evolved image filters using CGP (Sekanina, Vašíček).

# Other successful applications of GP

- Predictors, classifiers, schedulers, controllers, etc.
  - Not as scalable as DNNs, but the resulting solution can be optimized for error as well as the size and other criteria
  - Their behavior can often be explained.
- Game strategies
- Design of DNNs, circuits, regulators, antennas, quantum circuits, antennas, …
- Automated design of program tests
- Automated software repair
- Genetic improvement of software
- Automated patent invention



# Humies competition at GECCO conferences



Human-Competitive Results Produced By Genetic And Evolutionary Computation
http://www.human-competitive.org/

# Humies competition

J. Koza: We say that *an automatically created result is "human-competitive" if it satisfies one or more of the eight criteria below*.

- (A) The result was patented as an invention in the past, is an improvement over a patented invention, or would qualify today as a patentable new invention.
- (B) The result is equal to or better than a result that was accepted as a new scientific result at the time when it was published in a peer-reviewed scientific journal.
- (C) The result is equal to or better than a result that was placed into a database or archive of results maintained by an internationally recognized panel of scientific experts.
- (D) The result is publishable in its own right as a new scientific result ¾ *independent* of the fact that the result was mechanically created.
- (E) The result is equal to or better than the most recent human-created solution to a long-standing problem for which there has been a succession of increasingly better human-created solutions.
- (F) The result is equal to or better than a result that was considered an achievement in its field at the time it was first discovered.
- (G) The result solves a problem of indisputable difficulty in its field.
- (H) The result holds its own or wins a regulated competition involving human contestants (in the form of either live human players or human-written computer programs).

# Humies winners (2004 - 2020)

| Year | Humies Winners | Institution |
|---|---|---|
| 2004 | An Evolved Antenna for Deployment on NASA's ST 5 Mission | NASA AMES, US |
| | Automatic Quantum Computer Programming: A GP Approach | Hampshire Coll., US |
| 2005 | Two-dimensional photonic crystals designed by evolutionary algorithms | Cornell U., US |
| | Shaped-pulse optimization of coherent soft-x-rays | Colorado S. Univ., US |
| 2006 | Catalogue of Variable Freq. and Single-Resistance-Controlled Oscillators | MIT, US |
| 2007 | Evol. Design of Single-Mode Microstructured Polymer Optical Fibers | UCL, UK |
| 2008 | Genetic Programming for Finite Algebras | Hampshire Coll., US |
| 2009 | A Genetic Programming Approach to Automated Software Repair | U. of New Mexico, US |
| 2010 | Evol. design of the energy function for protein structure prediction | U. of Nottingham, UK |
| 2011 | GA-FreeCell: Evolving Solvers for the Game of FreeCell | Ben-Gurion U., IL |
| 2012 | Evolutionary Game Design | ICL, UK |
| 2013 | Evolutionary Design of FreeCell Solvers | Ben-Gurion U., IL |
| | Search for a grand tour of the Jupiter Galilean moons | ESA, EU |
| 2014 | Genetic Algorithms for Evolving Computer Chess Programs | Bar-Ilan University, IL |
| 2015 | Evolutionary Approach to Approximate Digital Circuits Design | Brno U. of Tech., CZ* |
| 2016 | Automated Software Transplantation | UCL, UK |
| 2017 | Explaining quantum correlations through evolution of causal models | U. of Sydney, AU |
| 2018 | A New EA-Based Home Monitoring Device for Parkinson's Dyskinesia | U. of York, UK |
| 2019 | Automated Self-Optimization in Heterogeneous Wireless Comm. Networks | UCD, IE |
| 2020 | Low cost satellite constellations for nearly continuous global coverage | Aerospace corporation, US |

FIT BUT – Silver medal (2008, 2011, 2016), Bronze medal (2014, 2018)

# References


Riccardo Poli, William B Langdon, Nicholas Freitag McPhee: A Field Guide to Genetic Programming, Lulu.com 2008, http://www.lulu.com/content/2167025

Koza J. et al.: Genetic Programming IV: Routine Human-Competitive Machine Intelligence, Springer, 2005

Koza J.R. Introduction to Genetic programming (tutorial), GECCO 2004

http://www.genetic-programming.com/johnkoza.html#anchor6005241

The Genetic Programming Bibliography at http://gpbib.cs.ucl.ac.uk/

Eiben A. E., Smith J.E.: Introduction to Evolutionary Computing, Springer, 2007

https://www.cs.vu.nl/~gusz/ecbook/ecbook.html

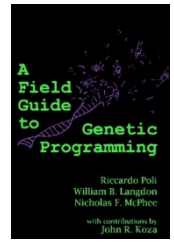
Miller J.F. Cartesian Genetic Programming, Springer, 2011

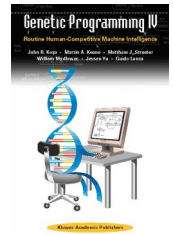
https://www.cartesiangp.com/

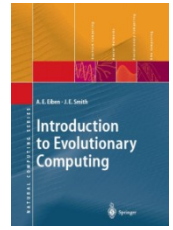
Sekanina L., Vašíček Z., Mrázek V.: Automated Search-Based Functional Approximation for Digital Circuits. Approximate Circuits - Methodologies and CAD. Springer, 2019, pp. 175-203

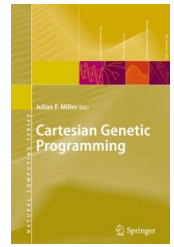

DEAP: Distributed Evolutionary Algorithms in Python

https://deap.readthedocs.io/en/master/



# Thank you for your attention!

Michal Bidlo

Faculty of Information Technology

Brno University of Technology

bidlom@fit.vut.cz

https://www.fit.vut.cz/person/bidlom/

BISSIT, Brno, July 22, 2022