BABEŞ-BOLYAI UNIVERSITY
Faculty of Computer Science and Mathematics

ARTIFICIAL INTELLIGENCE

Intelligent systems

Machine learning

Genetic Programming

Topics

A. Short introduction in Artificial Intelligence (AI)

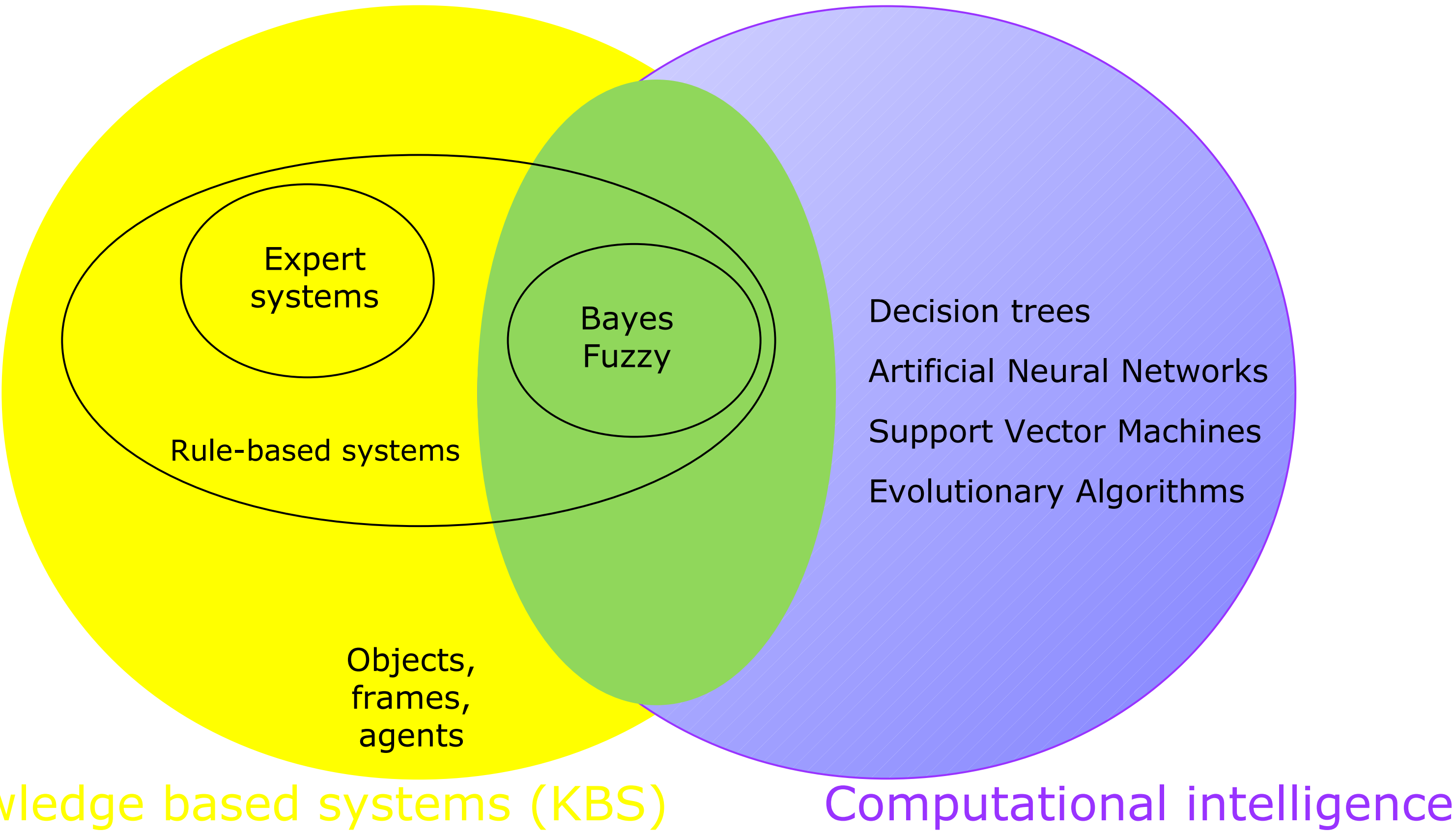
A. Solving search problems
   A. Definition of search problems
   B. Search strategies
      A. Uninformed search strategies
      B. Informed search strategies
      C. Local search strategies (Hill Climbing, Simulated Annealing, Tabu Search, Evolutionary algorithms, PSO, ACO)
      D. Adversarial search strategies

C. **Intelligent systems**
   A. Rule-based systems in certain environments
   B. Rule-based systems in uncertain environments (Bayes, Fuzzy)
   C. **Learning systems**
      A. Decision Trees
      B. Artificial Neural Networks
      C. **Evolutionary algorithms**
      D. Support Vector Machines
   D. Hybrid systems

Useful information

- Chapter 15 of *C. Groşan, A. Abraham, Intelligent Systems: A Modern Approach, Springer, 2011*
- Chapter 9 of *T. M. Mitchell, Machine Learning, McGraw-Hill Science, 1997*
- Documents from *12_svm* and *13_GP folders*

Intelligent systems



Intelligent systems – Machine Learning

- Typology
  - Experience criteria:
    - Supervised learning
    - Unsupervised learning
    - Active learning
    - Reinforcement learning
  - Algorithm criteria
    - Decision trees
    - Artificial Neural Networks
    - **Evolutionary Algorithms**
    - Support Vector Machines
    - Hidden Markov Models

Intelligent systems – machine learning

- Genetic programming (GP)
  - Definition
  - Design
  - Advantages
  - Limits
  - Versions

Intelligent systems - GP

Remember

- Supervised learning → regression problem (study of relations among variables)
  - For a set of n data (examples, instances, cases)
    - Training data – as pairs (($atribute\_data_i$, $output_i$), where
      - $i = 1,n$ (n = # of training data)
      - ***atribute_data**$_i$* = (atr_{i1}, atr_{i2}, ..., atr_{im}), m – # of attributes (characteristics, properties) of an example
      - *$output_i$* – a real number
    - Testing data
      - (***atribute_data**$_i$*), $i = n+1,N$ ($N-n$ = # of testing data)
  - Determine
    - An (unknown) function that maps the attributes info outputs on training data
    - Output (value) of a (new) test data by using the learnt function (on training data)

- How we find the function (expression)?
  - Evolutionary algorithms → genetic programming

Intelligent systems - GP

Remember

- Evolutionary algorithms
  - Nature-inspired (bio-inspired)
  - Iterative
  - Based on
    - Populations of potential solutions
    - Random search guided by
      - Natural selection operation
      - Crossover and mutation operations
  - Parallel processing of more solutions
- Evolutionary metaphor

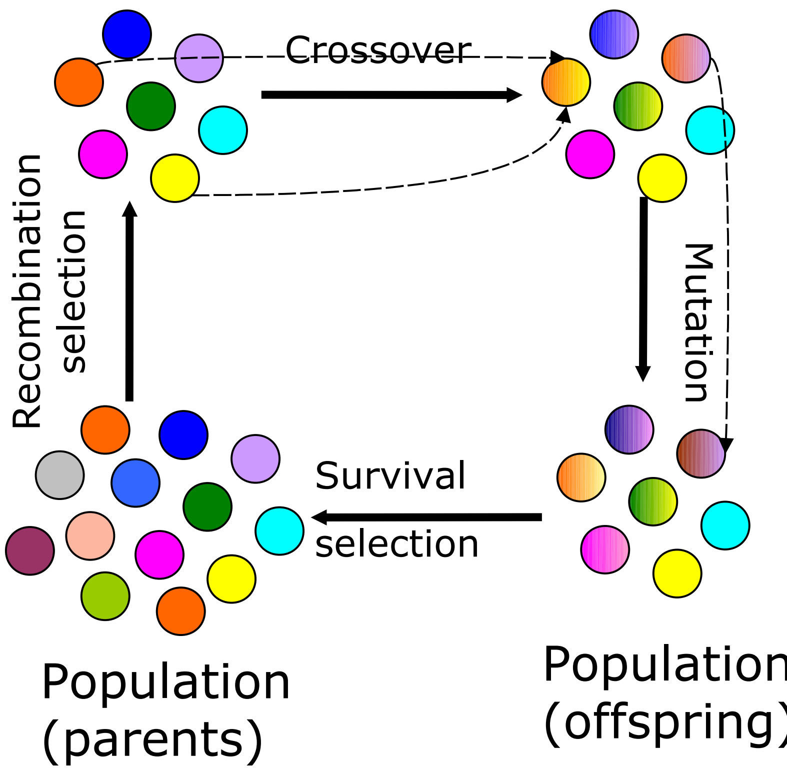
Natural evolution	Problem solving
Individual	Possible solution
Population	Set of possible solutions
Chromosome	Solution coding (representation)
Gene	Part of representation
Fitness (adaptation measure)	Quality
Crossover and mutation	Search operators
Environment	Problem search space

Intelligent systems - GP

Remember

- Evolutionary algorithm

```
Initialisation P(0)
Evaluation P(0)
g := 0; //generation
while (not stop_condition) execute
    Repeat
        Select 2 parents p1 and p2 from P(g)
        Crossover(p1,p2) ➔ o1 and o2
        Mutation(o1) ➔ o1*
        Mutation(o2) ➔ o2*
        Evaluation(o1*)
        Evaluation(o2*)
        Add o1* and o2* in P(g+1)
    until P(g+1) is complete
    g := g + 1
EndWhile
```

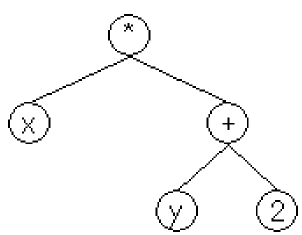


Intelligent systems - GP

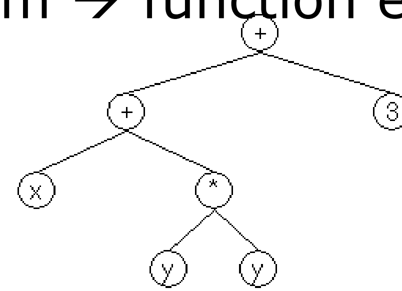
- Definition
  - Proposed by Koza in 1988
  - http://www.genetic-programming.org/
  - A special case of evolutionary algorithms
  - Chromosome
    - As trees that encode small programs
  - Fitness of a chromosome
    - Performance of the program encoded by the chromosome
  - GP's aim
    - Evolving computer programs
    - Gas evolve solutions for particular problems only

Intelligent systems - GP

- Design
  - Chromosome representation
    - Very important, but a difficult task
    - Chromosome = tree with nodes of type
      - Function → (mathematical) operators (+,-,*,/,*sin*,*log*, *if*,...)
      - Terminal → attributes of data or constants (*x*,*y*,*z*,*a*,*b*,*c*,...)
    - that encodes the mathematical expression of the program (regression problem → function expression)



$x(y+2)$



$x+y^2+3$

Intelligent systems - GP

- Design
  - Fitness
    - Prediction error – difference between what we want to obtain and what we actually obtain
    - For a regression problem with input data (2 attributes and an output) and 2 chromosomes:
      - $c_1 = 3x_1 - x_2 + 5$
      - $c_2 = 3x_1 + 2x_2 + 2$

$f^*(x1,x2) = 3x_1 + 2x_2 + 1$ – unknown

x_1	x_2	$f^*(x_1,x_2)$	$f_1(x_1,x_2)$	$f_2(x_1,x_2)$	$\lvert f^*-f_1\rvert$	$\lvert f^*-f_2\rvert$
1	1	6	7	7	1	1
0	1	3	4	4	1	1
1	0	4	8	5	4	1
-1	1	0	1	1	1	1
					Σ=7	Σ= 4

→ c_2 is better than c_1

Intelligent systems - GP

- Design
  - Fitness
    - Prediction error – difference between what we want to obtain and what we actually obtain
    - For a classification problem with input data (2 attributes and an output) and 2 chromosomes:
      - $c_1 = 3x_1 - x_2 + 5$
      - $c_2 = 3x_1 + 2x_2 + 2$

x_1	x_2	$f^*(x_1,x_2)$	$f_1(x_1,x_2)$	$f_2(x_1,x_2)$	$\|f^*-f_1\|$	$\|f^*-f_2\|$
1	1	Yes	Yes	Yes	0	0
0	1	No	Yes	No	1	0
1	0	Yes	No	No	1	1
-1	1	Yes	No	yes	1	0
					$\Sigma=3$	$\Sigma= 1$

→ c_2 is better than c_1

Intelligent systems - GP

- Design
  - Chromosome initialisation
    - Random generation of correct trees → valid programs (valid mathematical expressions)
    - Establish a maximal depth of the trees D_{max}
    - 3 initialisation methods
      - *Full* → each branch of the root has depth D_{max}
        - Nodes of depth $d < D_{max}$ are initialised by a function from F
        - Nodes of depth $d = D_{max}$ are initialised by a terminal from T
      - *Grow* → each branch of the root has a depth $< D_{max}$
        - Nodes of depth $d < D_{max}$ are initialised by an element from $F \cup T$
        - Nodes of depth $d = D_{max}$ are initialised by a terminal from T
      - *Ramped half and half* → ½ of population is initialised by using *full* method and ½ of population is initialised by *grow* method

Intelligent systems - GP

- Design
  - Genetic operators → recombination selection
    - Similar to other EAs
    - Advise → proportional selection
    - over-selection → for very large populations
      - Sort the population based on fitness and consider 2 groups:
        - Group 1 contains the best x% chromosomes from population
        - Group 2 contains (100-x)% chromosome from population
        - For populations with 1000, 2000, 4000 or 8000 chromosomes, x can be 32%, 16%, 8% and 4% respectively
      - 80% of selection operators will choose chromosomes from the first group and 20% of them from the second group

Intelligent systems - GP

- Design
  - Genetic operators → survival selection
    - Sketches
      - Generational
      - Steady-state
    - Problems
      - *Bloat* → the fattest individual survives (size of chromosomes increases during evolution)
      - Solutions
        - Block the variation operators that produce to fat offsprings
        - parsimony pressure - to give a penalty in the cost function (or fitness function) to long programs or program with many non-coding parts

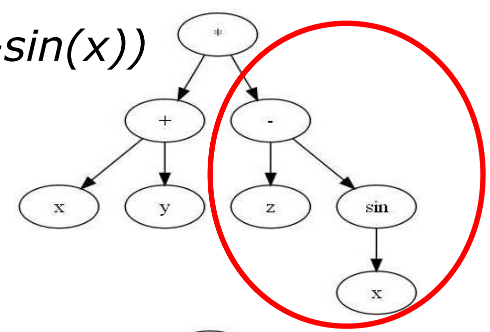
Intelligent systems - GP

- Design
  - Genetic operators → crossover and mutation
    - Parameters
      - A probability p of choosing between XO and mutation
        - p = 0 (cf. Koza) or p = 0.05 (cf. Banzhaf)
      - Two probabilities p_c and p_m for establishing the part of chromosome(s) that must be changed
    - Size of offsprings can be different to size of parents

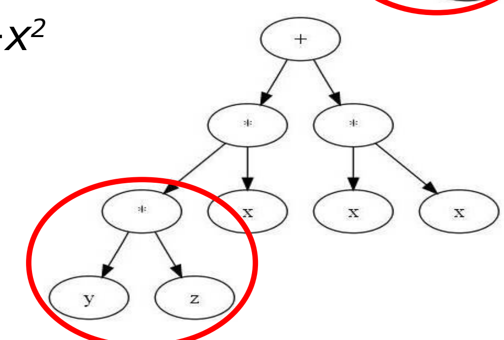
Intelligent systems - GP

- Design
  - Genetic operators → crossover
    - By cutting point
      - sub-trees are exchanged
      - Cutting point is randomly generated

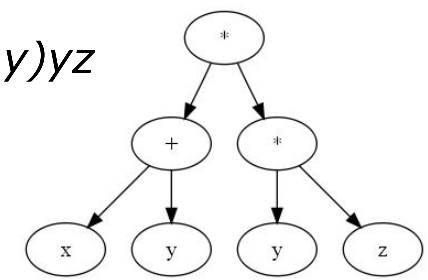
$p_1=(x+y)*(z-\sin(x))$



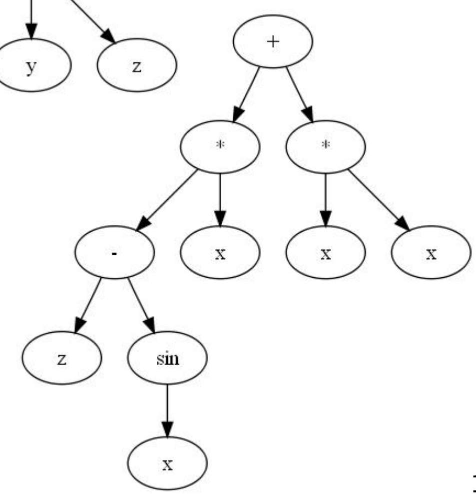
$p_2=xyz+x^2$



$f_1=(x+y)yz$



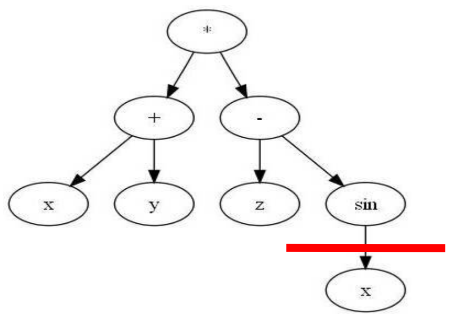
$f_2=(z-\sin(x))x+x^2$



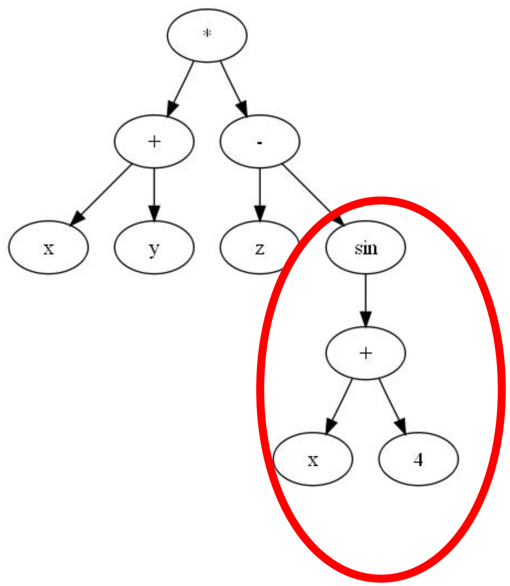
Intelligent systems - GP

- Design
  - Genetic operators → mutation
    - *Grow* mutation → replace a leaf by a new sub-tree

p=(x+y)(z-sin(x))*



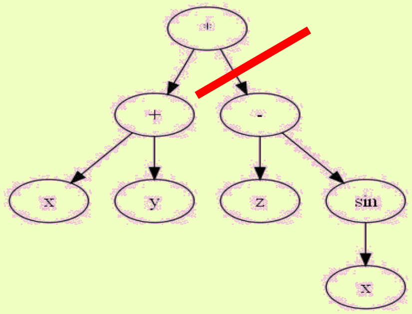
f=(x+y)(z-sin(x+4))*



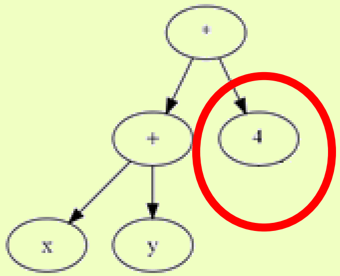
Intelligent systems - GP

- Design
  - Genetic operators → mutation
    - *Shrink* mutation → replace a sub-tree by a leaf

p=(x+y)(z-sin(x))*



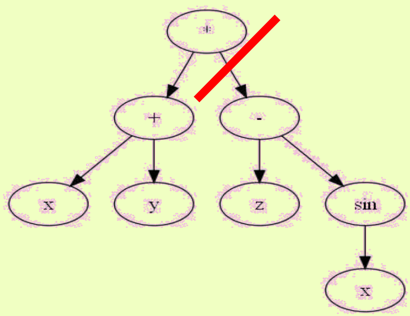
*f=(x+y)*4*



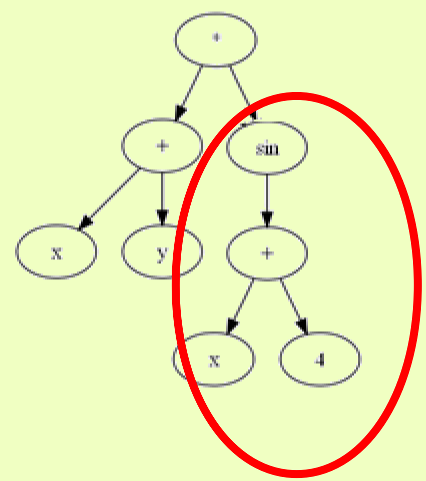
Intelligent systems - GP

- Design
  - Genetic operators → mutation
    - *Koza* mutation → replace a node (leaf or internal) by a sub-tree

p=(x+y)(z-sin(x))*



*f=(x+y)*sin(x+4)*



Intelligent systems - GP

- Design
  - Genetic operators → mutation
    - *Switch* mutation
      - Select an internal node and re-order its sub-trees
    - *Cycle* mutation
      - Select a node and replace it by a new node of the same type (internal node with a function, leaf node with a terminal)

Intelligent systems - GP

- GAs vs. GP
  - Chromosome’s shape
    - GAs – linear chromosomes
    - GP – non-linear chromosomes
  - Chromosome’s size
    - GAs – fix size
    - GP – variable size (depth or width)
  - Offspring generation
    - GAs – XO and mutation
    - GP – XO or mutation

Intelligent systems - GP

- Advantages
  - GP finds solutions for problems without an optimal solution
    - A program for car driving → there are more solution
      - Some solutions → safe but slow driving
      - Other solutions → dangerous, but fast driving
      - Car driving ←→ trade-off large speed and safety
    - GP is useful for problems whose variables are frequently changed
      - Car driving on a highway
      - Car driving on a forest road
- Limits
  - Large time required for evolving the solution

Intelligent systems - GP

- GP versions
  - Linear GP (Cramer, Nordin)
  - Gene Expression Programming (Ferreira)
  - Multi Expression Programing (Oltean)
  - Gramatical Evolution (Ryan, O'Neill)
  - Cartesian Genetic Programming (Miller)

Intelligent systems - GP

- Linear GP
  - Evolving programs written in an imperative language (fitness computation does not require interpretation) → works fast
  - Representation
    - Vector of statements, each statement being (in the case of a maximal arrity of n for a function)
      - Index_op, out_register, in_1_register, in_2_register,…, in_n_register

- $v_i = v_j * v_k$ // instruction operating on two registers
- $v_i = v_j * c$ // instruction operating on one register and one constant
- $v_i = sin(v_j)$ // instruction operating on one register

```
void LGP_program (double v[11])
  {
      ...
      v[8] = v[0] - 10;
      v[6] = v[2] * v[0];
      v[5] = v[8] * 7;
      v[4] = v[2] - v[0];
      v[10] = v[1]/v[4];
      v[3] = sin(v[1]);
      v[1] = v[8] - v[6];
      v[7] = v[10] * v[3];
      v[9] = v[0] + v[7];
      v[2] = v[7] + 3;
      ...
  }
```

```
void LGP_effective_program (double v[11])
  {
      ...
      v[4] = v[2] - v[0];
      v[10] = v[1]/v[4];
      v[3] = sin(v[1]);
      v[7] = v[10] * v[3];
      v[9] = v[0] + v[7];
      ...
  }
```

Intelligent systems - GP

- Linear GP
  - Initialisation
    - Random
    - Constraints
      - Initial length of chromosome (# of statements)
  - Variation genetic operators
    - Crossover – 2-cutting points
    - Mutation
      - Micro-mutation → change an operand or operator (without modifying the size of chromosome)
      - Macro-mutation → insert or eliminate a statement (modifying the size of chromosome)

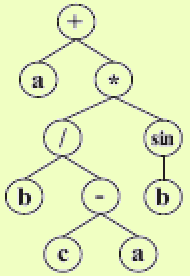
Intelligent systems - GP

- Linear GP
  - Advantages
    - Evolution into a low-level language
  - Disadvantages
    - # of required registers (# of problem’s attributes)
  - Resources
    - Register Machine Learning Technologies http://www.aimlearning.com
    - *Peter Nordin's home page http://fy.chalmers.se/~pnordin*
    - *Wolfgang Banzhaf's home page http://www.cs.mun.ca/~banzhaf*
    - *Markus Brameier's home page http://www.daimi.au.dk/~brameier*

Intelligent systems - GP

- Gene Expression Programming (GEP)
  - Main idea
    - Linear representation of expressions that can be encoded in a tree (by breadth-first traversing procedure)

$$C = +a * /Sb - bcacabbc$$



  - Representation
    - A chromosome is composed by more genes
      - Linked by + or *
    - Each gene is composed by
      - Head
        - Contains functions and terminals
      - Tail
        - Contains t terminals only, where t = (n-1)*h+1, with n – maximal arrity of a function from F

Intelligent systems - GP

- GEP
  - Initialisation
    - Randomly, by elements from F and T (cf. to previous rules)
  - Variation genetic operators
    - Crossover
      - At allele level
        - One cutting point
        - Two cutting points
      - At gene level
        - Chromosomes exchange (between them) some genes (located on corresponding positions)
    - Mutation
      - At allele level
        - Change an element from head or tail (following the previous initialisation rules)
    - Transpositions
      - the introduction of an insertion sequence somewhere in a chromosome

Intelligent systems - GP

- GEP
  - Advantages
    - Coding into chromosomes some correct programs due to gene splitting in head and tail
  - Disadvantages
    - Multi-gene chromosomes
      - How many genes?
      - How to link the genes?
  - Resources
    - Gene Expression Programming website, http://www.gepsoft.com
    - *Heitor Lopes's home page* *http://www.cpgei.cefetpr.br/~hslopes/index-english.html*
    - *Xin Li's home page* *http://www.cs.uic.edu/~xli1*
    - *GEP in C#* *http://www.c-sharpcorner.com/Code/2002/Nov/GEPAlgorithm.asp*

Intelligent systems - GP

- Multi Expression Programming (MEP)
  - Main idea
    - Chromosome is composed by more genes, each gen being a 3 address code
      - Similarly to GEP, but faster
  - Representation
    - Linear
    - A gene contains a (binary or unary) function and pointers to its arguments
    - Chromosome encodes more possible solutions → each solution corresponds to a gene
      - Quality of a solution (gene) = sum (over training data) of differences between what we want to obtain and what we obtain
      - Fitness = quality of the best gene

Intelligent systems - GP

- MEP
  - Initialisation
    - First gene must be a terminal
    - Other genes can contain
      - A terminal or
      - A (unary or binary) function and pointers to its arguments
        - Arguments of a function located in the i^{th} gene must be located in genes of index < I
  - Variation genetic operators
    - Crossover → exchange some genes between parents
      - 1-cutting point
      - 2-cutting points
      - Uniform
    - Mutation → modify a gene
      - First gene → randomly generate a new terminal
      - Other genes → randomly generate a terminal or a function (function symbol and its arguments)

Intelligent systems - GP

- MEP
  - Advantages
    - Dynamic output for each chromosome
      - Complexity of the search program (expression)
      - Programs (expression) of variable length obtained without special operators
      - Program of exponential length encoded into chromosomes of polynomial length
  - Disadvantages
    - Complexity of decoding for unknown training data → evolving game's strategies
  - Resources
    - Mihai Oltean's home page http://www.cs.ubbcluj.ro/~
    - *Crina Gro»san's home page http://www.cs.ubbcluj.ro/~cgrosan*
    - MEP web page http://www.mep.cs.ubbcluj.ro
    - *MEP in C# http://www.c-sharpcorner.com*

Intelligent systems - GP

- Grammatical Evolution (GE)
  - Main idea
    - Evolving programs in Backus-Naur form (program expressed as a grammar with terminal symbols, non-terminals, start symbol and rules)
  - Representation
    - Binary strings of codons (groups of 8 bits) → rule that must be applied
    - Example
      - G={N,T,S,P}, N={+, -,*, /, sin, (,)}, T= {expr, op2, op1}, S=⟨expr⟩, P is:
        - ⟨expr⟩ ::=a|b|c| ⟨expr⟩ ⟨op2⟩ ⟨expr⟩|(⟨expr⟩ ⟨op2⟩ ⟨expr⟩)| ⟨op1⟩ ⟨expr⟩
        - ⟨op2⟩::=+|-|*|/,
        - ⟨op1⟩::=sin
      - C^*_{GE}=(9 12 12 3 15 7 11 4 2 5 0 6 11 0 1 7 12)
      - S= ⟨expr⟩ ➔ ⟨expr⟩ ⟨op2⟩ ⟨expr⟩ ➔ a⟨op2⟩ ⟨expr⟩ ➔a + ⟨expr⟩ ➔ a + ⟨expr⟩ ⟨op2⟩ ⟨expr⟩ ➔ a + ⟨expr⟩ ⟨op2⟩ ⟨expr⟩ ⟨op2⟩ ⟨expr⟩ ➔ a + b ⟨op2⟩ ⟨expr⟩ ⟨op2⟩ ⟨expr⟩

$$
\begin{aligned}
C_{GE} = (&00001001\ 00001100\ 00001100\ 00000011\ 00001111\ 00000111\\
&00001011\ 00000100\ 00000010\ 00000101\ 00000000\ 00000110\\
&00001011\ 00000000\ 00000001\ 00000111\ 00001100)
\end{aligned}
$$

$$
\begin{aligned}
&a+b/\langle expr\rangle\langle op_2\rangle\langle expr\rangle\\
&a+b/(\langle expr\rangle\langle op_2\rangle\langle expr\rangle)\langle op_2\rangle\langle expr\rangle\\
&a+b/(c\langle op_2\rangle\langle expr\rangle)\langle op_2\rangle\langle expr\rangle\\
&a+b/(c-\langle expr\rangle)\langle op_2\rangle\langle expr\rangle\\
&a+b/(c-a)\langle op_2\rangle\langle expr\rangle\\
&a+b/(c-a)*\langle expr\rangle\\
&a+b/(c-a)*\langle op_1\rangle\langle expr\rangle\\
&a+b/(c-a)*\sin\langle expr\rangle
\end{aligned}
$$

$$E = a + b/(c-a)*\sin(b)$$

Intelligent systems - GP

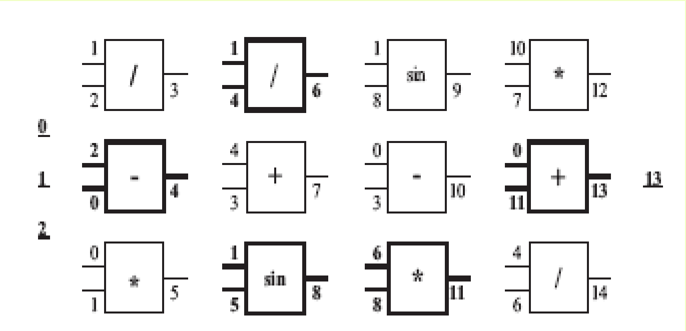
- GE
  - Initialisation
    - Binary string is randomly initialised by 0 or 1 (without constraints) → valid programs
    - Decoding ends when a complete program is obtained
      - If the codons end and the program is incomplete, restart the codons from beginning → wrapping
  - Variation genetic operators
    - Crossover
      - Cutting point XO
    - Mutation
      - Probabilistic change of a bit into its opponent
    - Duplication
      - A sequence of genes is copied to the end of chromosome
    - Pruning
      - Elimination of unused genes

Intelligent systems - GP

- GE
  - Advantages
    - Evolving programs written in languages whose statements can be expressed as BNF rules
    - Representation can be changed by changing the grammar
  - Disadvantages
    - Infinite wrapping → limit the repetitions and penalise the chromosomes that overpass a given threshold of repetitions
  - Resources
    - Grammatical Evolution web page, http://www.grammatical-evolution.org
    - *Conor Ryan's home page, http://www.csis.ul.ie/staff/conorryan*
    - *Michael O'Neill's home page, http://ncra.ucd.ie/members/oneillm.html*
    - *John James Collins's home page, http://www.csis.ul.ie/staff/jjcollins*
    - *Maarten Keijzer's home page, http://www.cs.vu.nl/~mkeijzer*
    - *Anthony Brabazon's home page http://ncra.ucd.ie/members/brabazont.html*

Intelligent systems - GP

- Cartesian Genetic Programming (CGP)
  - Main idea
    - Chromosomes as graphs (matrix) → more complex programs
  - Representation
    - Cartesian system (matrix of nodes)
    - A node has associated
      - A function
      - Inputs
      - Outputs
    - Chromosome output
      - Output of any node



$$C = (1,2,3,\ \mathbf{2,0,1},\ 0,1,2,\ \mathbf{1,4,3},\ 4,3,0,\ \mathbf{1,5,4},\ 1,8,4,\ 0,3,1,\ \mathbf{6,8,2},\ 10,7,2,\ \mathbf{0,11,0},\ 4,6,3,\ 13)$$

Intelligent systems - GP

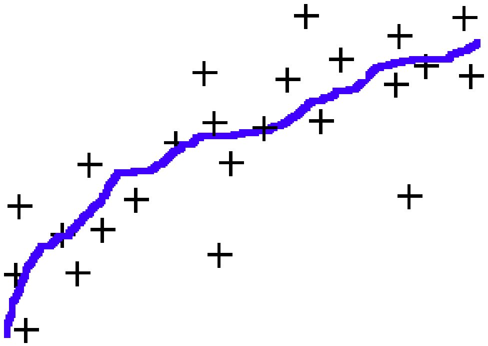
- CGP
  - Initialisation
    - Randomly
    - Inputs of any node must be nodes from previous columns
      - Nodes of the first column has as inputs the problem attributes
  - Variation genetic operators
    - Crossover
      - Is not applied
    - Mutation
      - Modify the elements of a node

Intelligent systems - GP

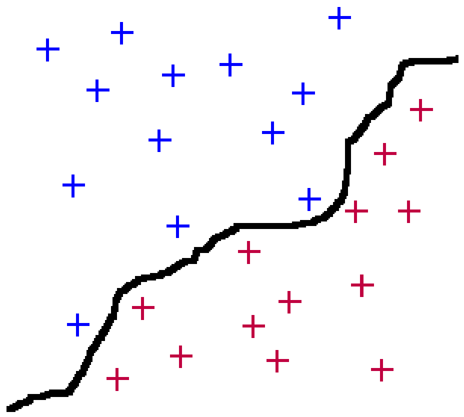
- CGP
  - Advantages
    - Evolving the index of node that provides the output of the program encoded into the chromosome
    - Evolved program can have one or more outputs
  - Disadvantages
    - number of columns influences the results
  - Resources
    - Julian. F. Miller's home page http://www.elec.york.ac.uk/intsys/users/jfm7
    - *Lukás Sekanina's home page http://www.fit.vutbr.cz/~sekanina/*

Intelligent systems - GP

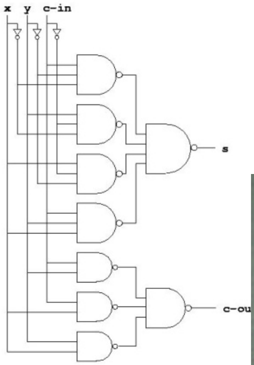
- Applications
  - Problems with relations between inputs and outputs
  - Regression problems



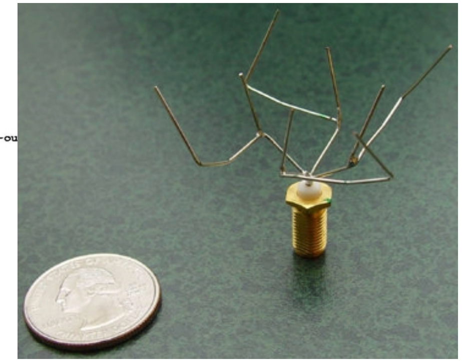
  - Classification problems



Intelligent systems - GP



- Applications
  - Design problems
    - Evolving digital circuits



    - Evolving antenna
      - http://idesign.ucsc.edu/projects/evo_antenna.html
    - Evolving programs

    - Evolving pictures and music
      - http://www.cs.vu.nl/~gusz/
    - Others
      - http://www.genetic-programming.com/humancompetitive.

Review

- Machine learning
  - Genetic programming (GP)
    - Evolutionary algorithms with chromosomes as trees
    - Chromosomes
      - Trees
      - Matrix
      - Linear
    - Encode potential solutions
      - Mathematical expressions → regression/classification problems
      - Boolean expressions → Even Parity problems or digital circuits design
      - Programs → evolving source codes for problem solving

Next lecture

A. Short introduction in Artificial Intelligence (AI)

A. Solving search problems
   A. Definition of search problems
   B. Search strategies
      A. Uninformed search strategies
      B. Informed search strategies
      C. Local search strategies (Hill Climbing, Simulated Annealing, Tabu Search, Evolutionary algorithms, PSO, ACO)
      D. Adversarial search strategies

C. **Intelligent systems**
   A. Rule-based systems in certain environments
   B. Rule-based systems in uncertain environments (Bayes, Fuzzy)
   C. **Learning systems**
      A. Decision Trees
      B. Artificial Neural Networks
      C. Evolutionary algorithms
      D. **Support Vector Machines**
   D. Hybrid systems

Next lecture – useful information

- Chapter 15 of *C. Groşan, A. Abraham, Intelligent Systems: A Modern Approach, Springer, 2011*

- Chapter 9 of *T. M. Mitchell, Machine Learning, McGraw-Hill Science, 1997*

- Documents from *svm* folder

- Presented information have been inspired from different bibliographic sources, but also from past AI lectures taught by:
  - PhD. Assoc. Prof. Mihai Oltean – www.cs.ubbcluj.ro/~moltean
  - PhD. Assoc. Prof. Crina Groşan - www.cs.ubbcluj.ro/~cgrosan
  - PhD. Prof. Horia F. Pop - www.cs.ubbcluj.ro/~hfpop