Evolutionary Algorithms: Genetic Programming

Jiří Kubalík
Department of Cybernetics, CTU Prague

http://cw.felk.cvut.cz/doku.php/courses/a4m33bia/start

Genetic Programming (GP)

GP shares with GA the philosophy of survival and reproduction of the fittest and the analogy of naturally occurring genetic operators.

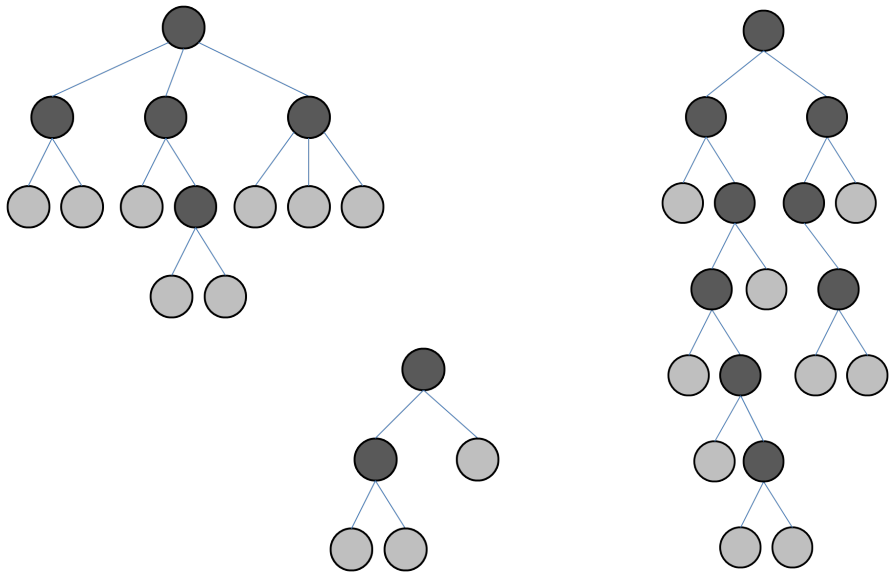
GP differs from GA in a representation, genetic operators and a scope of applications.

Structures evolved in GP are trees of dynamically **varying size and shape** representing hierarchical computer programs.

Chromosome of fixed length:

v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9	v_{10}

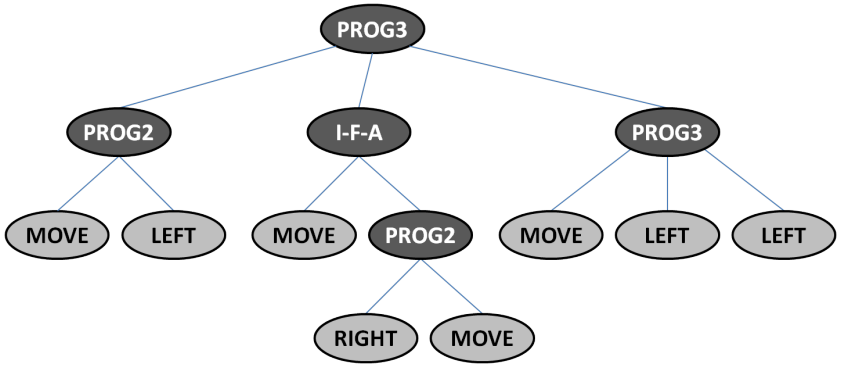
GP trees:



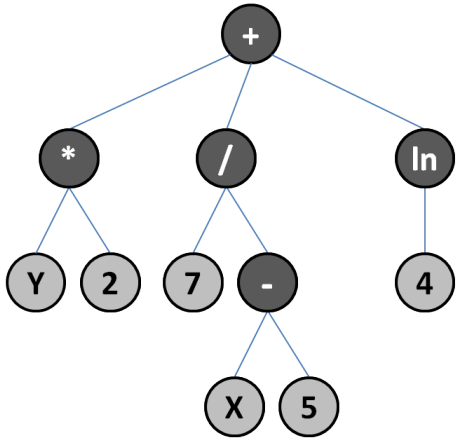
Genetic Programming (GP): Application Domains

Applications

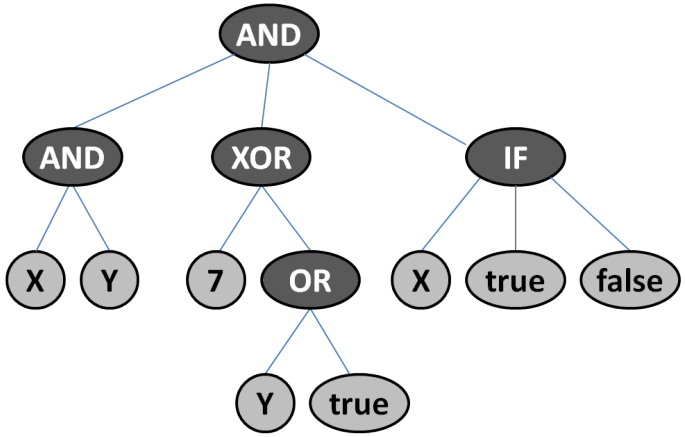
- learning programs,
- learning decision trees,
- learning rules,
- learning strategies,
- ...



Strategy for artificial ant



Arithmetic expression



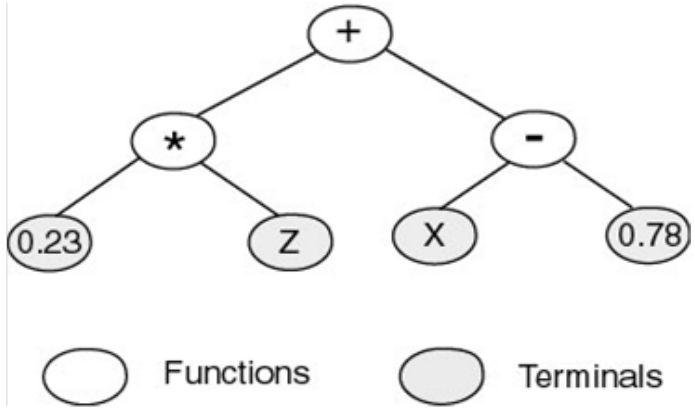
Logic expression

GP: Representation

All possible trees are composed of **functions** (inner nodes) and **terminals** (leaf nodes) appropriate to the problem domain

- **Terminals** – inputs to the programs (independent variables), real, integer or logical constants, actions.
- **Functions**
  - arithmetic operators (+, -, *, /),
  - algebraic functions (sin, cos, exp, log),
  - logical functions (AND, OR, NOT),
  - conditional operators (If-Then-Else, cond?true:false),
  - and others.

Example: Tree representation of a LISP S-expression $0.23 * Z + X - 0.78$



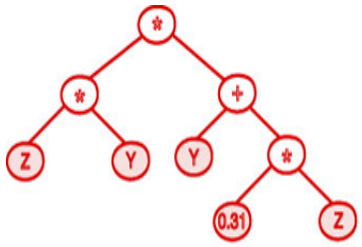
GP: Crossover

Subtree crossover

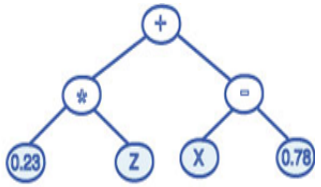
1. randomly select a node (crossover point) in each parent tree,
2. create the offspring by replacing the subtrees rooted at the crossover nodes.

Crossover points do not have to be selected with uniform probability

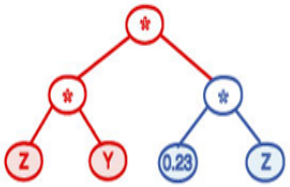
- Typically, the majority of nodes in the trees are leaves, because the average branching factor (the number of children of each node) is ≥ 2.
- To avoid swapping leave nodes most of the time, the widely used crossover scenario chooses function nodes 90% of the time and leaves 10% of the time.



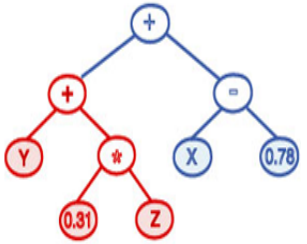
Parent 1: Z * Y * (Y + 0.31 * Z)



Parent 2: 0.23 * Z + X - 0.78



Child 1: 0.23 * Y * Z^2

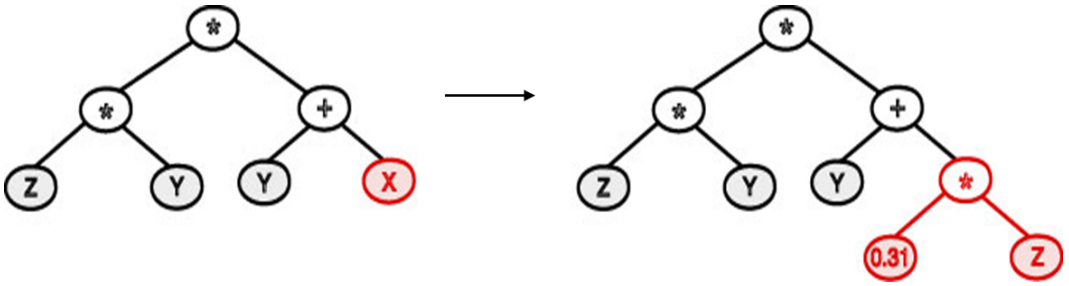


Child 2: Y + 0.31 * Z + X - 0.78

GP: Mutation

Subtree mutation

1. randomly select a mutation point from the set of all nodes in the parent tree,
2. substitute the subtree rooted at the picked node with the new subtree generated in the same way as used for generating trees for the initial population.



Point mutation

1. randomly select a mutation point from the set of all nodes in the parent tree,
2. substitute the primitive stored in the selected node with a different primitive of the same arity taken from the primitive set.

GP: Primitives sets properties

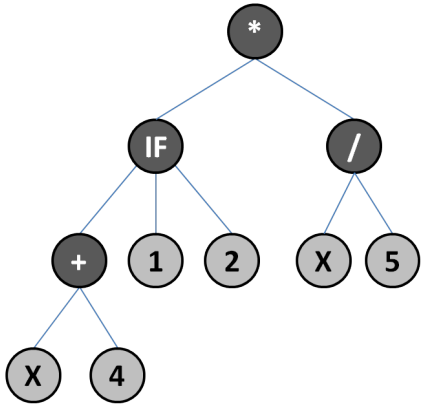
Closure

- *Type consistency* – any function and terminal can be used in any of the argument positions for every function in the function set.

  Usually, all the functions are return values of the same type, and each of their arguments have this type.

  Automatic conversion mechanism – a mechanism for converting values of one type to values of the required type. For example, converting real numbers to Booleans can be realized by treating all negative values as false and non-negative values as true.

  An alternative to ensure type consistency is *Strongly Typed GP*.



- *Evaluation safety* – to ensure that the GP computation does not fail at run time.

  Protected functions – first test for potential problems with its inputs before executing the corresponding instruction. If a problem is spotted then some default value is returned or a default action is taken.

  - Protected division – if the second argument is zero, then returns typically value 1.
  - Protected `MOVE_AHEAD` instruction – if a forward move is illegal then the instruction does nothing.

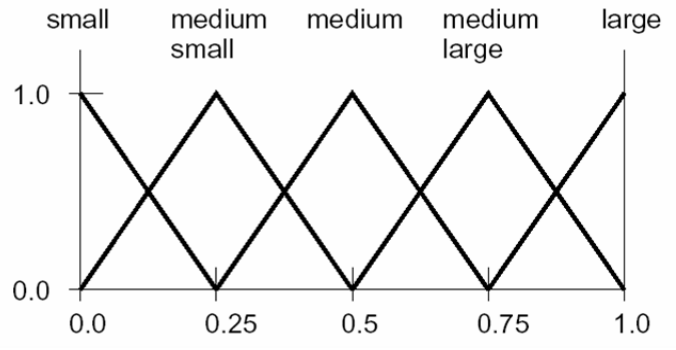
GP: Evolving Fuzzy-rule based Classifier

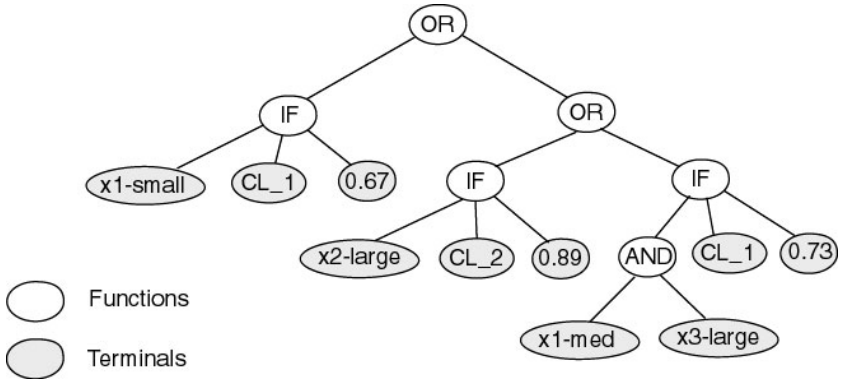
Classifier consists of fuzzy if-then rules of type

$$\text{IF } (x1 \text{ is } medium) \text{ and } (x3 \text{ is } large) \text{ THEN } class = 1 \text{ with } cf = 0.73$$

Linguistic terms – small, medium small, medium, medium large, large.

Fuzzy membership functions – approximate the confidence in that the crisp value is represented by the linguistic term.



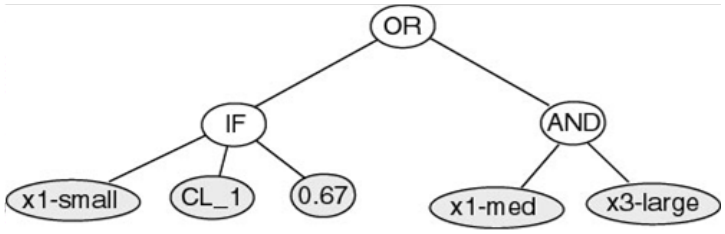


Three rules connected by OR

Illegal Tree

Blind **crossover and mutation operators can produce incorrect trees** that do not represent valid rule base.

- Obviously due to the fact that the **closure** property does not hold here.



What can we do?

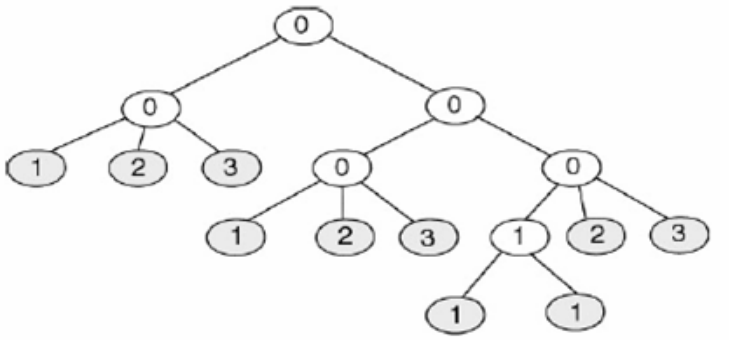
Strongly typed GP

Strongly typed GP – crossover and mutation make explicitly use of type information:

- every terminal has a type,
- every function has types for each of its arguments and a type for its return value,
- the genetic operators are implemented so that they do not violate the type constraints $\Longrightarrow$ only type correct solutions are generated.

Example: Given the representation specified as specified below, consider that we chose `IS` node (with return type 1) as a crossing point in the first parent. Then, the crossing point in the second parent must be either `IS` or `AND` node.

F / T	Output	Input
OR	0	0, 0
IF	0	1, 2, 3
AND	1	1, 1
IS	1	None
CLASS	2	None
CF	3	None



STGP can be extended to more complex type systems – multi-level and polymorphic higher-order type systems.

GP: Primitives sets properties

Sufficiency – it is possible to express a solution to the problem at hand using the elements of the primitive set.

Example: A sufficient primitive set for Boolean induction is {AND, OR, NOT, x_1, x_2, ..., x_n}.

When a primitive set is insufficient, GP can only develop programs that approximate the desired solution.

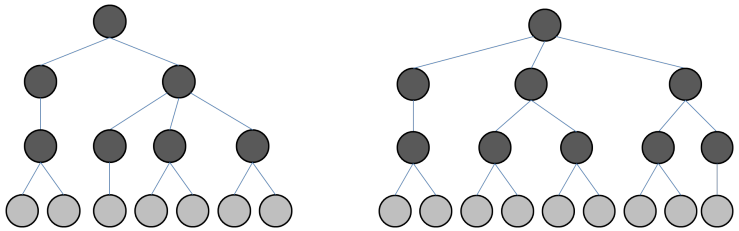
Example: {$+$, -, *, $/$, x, 0, 1} is insufficient primitive set for evolving transcendental function.

GP: Initialisation

D_{max} is the maximum initial depth of trees.

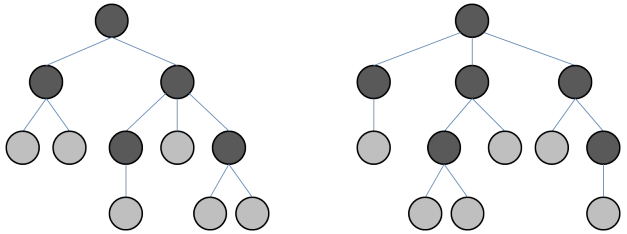
Full method(each branch has $depth = D_{max}$)

- nodes at depth $d < D_{max}$ randomly chosen from function set F,
- nodes at depth $d = D_{max}$ randomly chosen from terminal set T.



Grow method (each branch has $depth \leq D_{max}$)

- nodes at depth $d < D_{max}$ randomly chosen from $F \cup T$,
- nodes at depth $d = D_{max}$ randomly chosen from T.



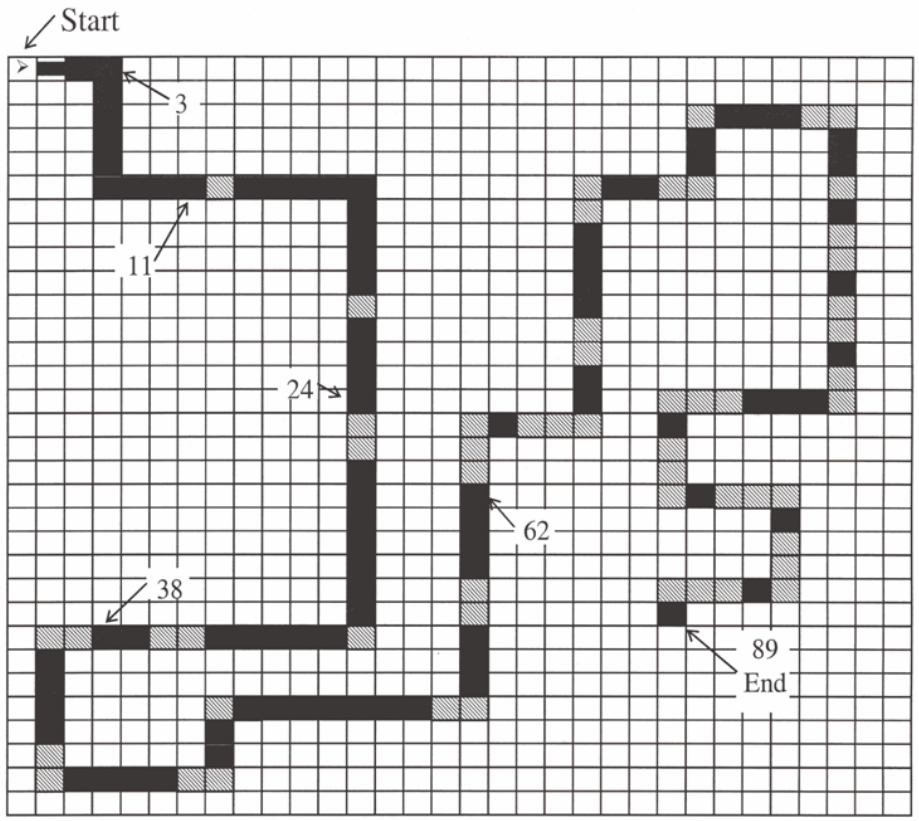
Ramped half-and-half – grow & full method each deliver half of initial population.

- A range of depth limits is used to ensure that trees of various sizes and shapes are generated.

Artificial Ant Problem

Santa Fe trail

- 32×32 **grid** with 89 food pieces.
- **Obstacles**
  - $1\times, 2\times$ strait,
  - $1\times, 2\times, 3\times$ right/left.



Ant capabilities

- **detects** the food right in front of him in direction he faces.
- **actions** observable from outside
  - MOVE – makes a step and eats a food piece if there is some,
  - LEFT – turns left,
  - RIGHT – turns right,
  - NO-OP – no operation.

Goal is to find a strategy that would navigate an ant through the grid so that it finds all the food pieces in the given time (600 time steps).

Artificial Ant Problem: GP Approach

Terminals

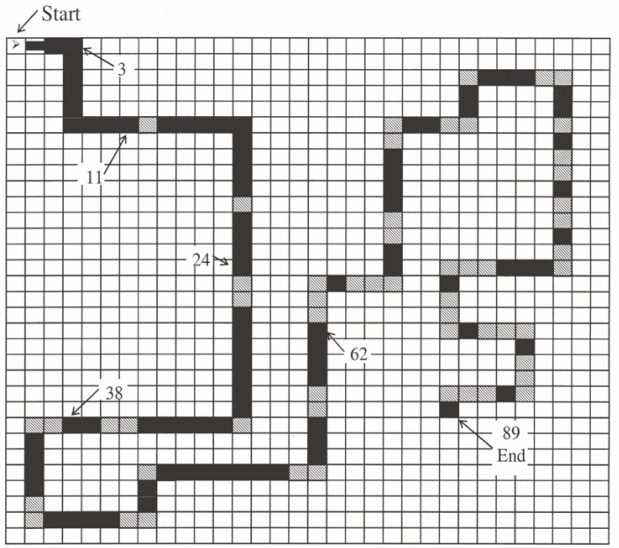
- motorial section,
- T = MOVE, LEFT, RIGHT.

Functions

- conditional IF-FOOD-AHEAD – food detection, 2 arguments (is/is_not food ahead),
- unconditional PROG2, PROG3 – sequence of 2/3 actions.

Ant repeats the program until time runs out (600 time steps) or all the food has been eaten.

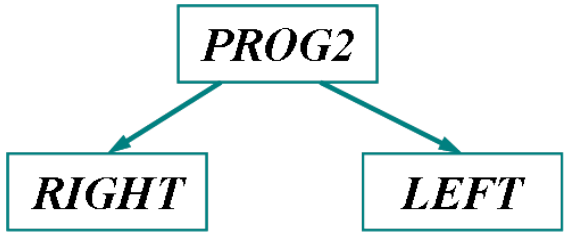
Santa Fe trail



Artificial Ant Problem: GP Approach cont.

Typical solutions in the initial population

- this solution



  completely fails in finding and eating the food,

- similarly this one

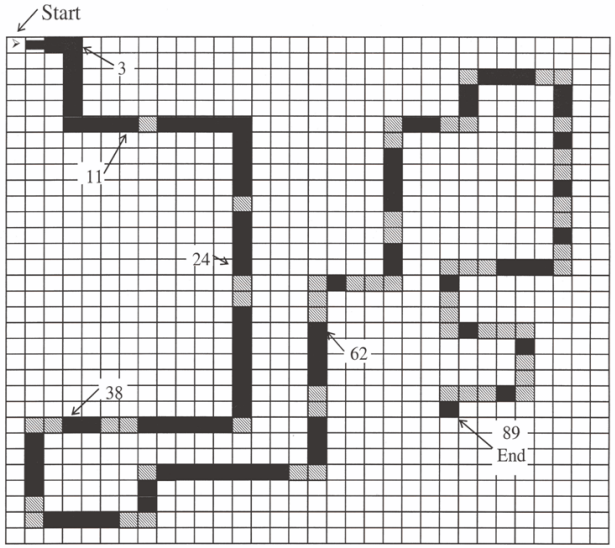
  (IF-FOOD-AHEAD (LEFT)(RIGHT)),

- this one

  (PROG2 (MOVE) (MOVE))

  just by chance finds 3 pieces of food.

Santa Fe trail



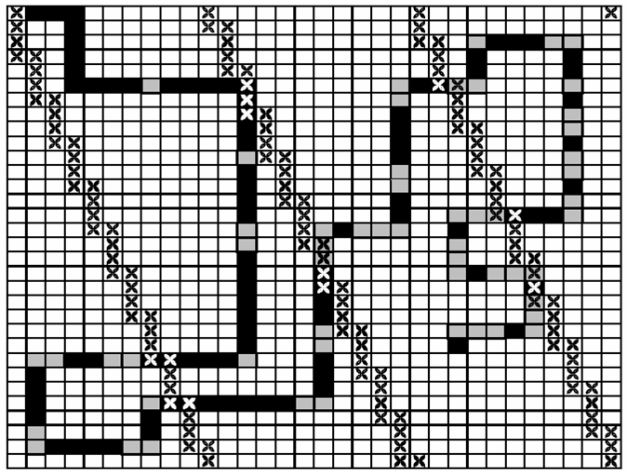
Artificial Ant Problem: GP Approach cont.

More interesting solutions

- **Quilter** – performs systematic exploration of the grid,

```
(PROG3  (RIGHT)
        (PROG3  (MOVE) (MOVE) (MOVE))
        (PROG2  (LEFT) (MOVE)))
```

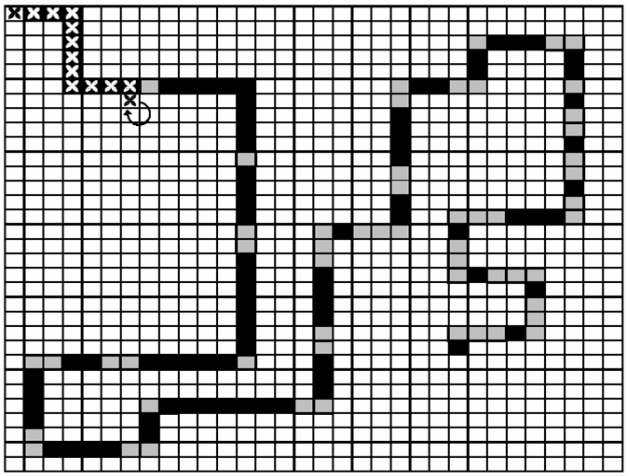
Quilter performance



- **Tracker** – perfectly tracks the food until the first obstacle occurs, then it gets trapped in an infinite loop.

```
(IF-FOOD-AHEAD (MOVE) (RIGHT))
```

Tracker performance

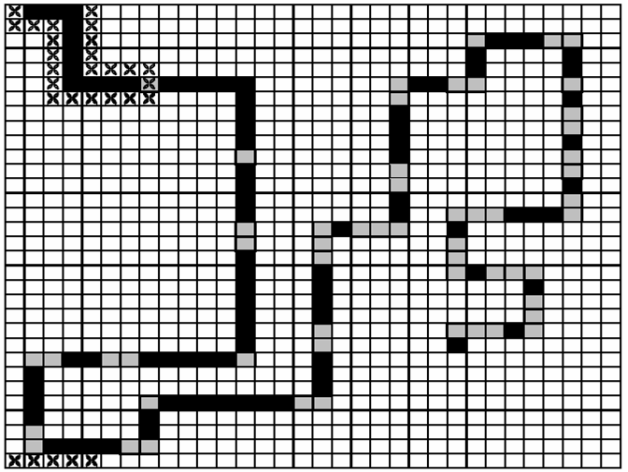


Artificial Ant Problem: GP Approach cont.

- **Avoider** – perfectly avoids food!!!

```
(I-F-A  (RIGHT)
        (I-F-A      (RIGHT)
                    (PROG2 (MOVE) (LEFT))))
```

Avoider performance



Average fitness in the initial population is 3.5

Artificial Ant Problem: GP result

In generation 21, the following solution was found that already navigates an ant so that he eats all 89 food pieces in the given time.

```
(I-F-A  (MOVE)
        (PROG3  (I-F-A      (MOVE)
                            (RIGHT)
                (PROG2      (RIGHT)
                            (PROG2      (LEFT)
                                        (RIGHT))))
                (PROG2      (I-F-A      (MOVE)
                                        (LEFT))
                            (MOVE))))
```

This program solves every trail with the obstacles of the same type as occurs in Santa Fe trail.

Compare the computational complexity with the GA approach!!!

GA approach: $65.536 \times 200 = 13 \times 10^6$ trials.

vs.

GP approach: $500 \times 21 = 10.500$ trials.

GP: Trigonometric Identity

Task is to find an equivalent expression to $cos(2x)$.

GP implementation:

- **Terminal set** $T = \{x, 1.0\}$.
- **Function set** $F = \{+, -, *, \%, sin\}$.
- **Training cases**: 20 pairs (x_i, y_i), where x_i are values evenly distributed in interval $(0, 2\pi)$.
- **Fitness**: Sum of absolute differences between desired y_i and the values returned by generated expressions.
- **Stopping criterion**: A solution found that gives the error less than 0.01.

Example of GP in Action: Trigonometric Identity cont.

1. run, 13^{th} **generation**

```
(- (- 1 (* (sin x) (sin x))) (* (sin x) (sinx)))
```

which equals (after editing) to $1 - 2 * sin^2 x$.

2. run, 34^{th} **generation**

```
(- 1 (* (* (sin x) (sin x)) 2))
```

which is just another way of writing the same expression.

Example of GP in Action: Trigonometric Identity cont.

1. run, 13^{th} **generation**

```
(- (- 1 (* (sin x) (sin x))) (* (sin x) (sinx)))
```

which equals (after editing) to $1 - 2 * sin^2 x$.

2. run, 34^{th} **generation**

```
(- 1 (* (* (sin x) (sin x)) 2))
```

which is just another way of writing the same expression.

3. run, 30^{th} **generation**

```
(sin (- (- 2 (* x 2))
        (sin (sin (sin (sin (sin (sin (* (sin (sin 1))
                                         (sin (sin 1))
                                         ))))))))
```

Example of GP in Action: Trigonometric Identity cont.

1. run, 13^{th} **generation**

```
(- (- 1 (* (sin x) (sin x))) (* (sin x) (sinx)))
```

which equals (after editing) to $1 - 2 * sin^2 x$.

2. run, 34^{th} **generation**

```
(- 1 (* (* (sin x) (sin x)) 2))
```

which is just another way of writing the same expression.

3. run, 30^{th} **generation**

```
(sin (- (- 2 (* x 2))
        (sin (sin (sin (sin (sin (sin (* (sin (sin 1))
                                         (sin (sin 1))
                                         )))))))))
```

(2 minus the expression on the 2nd and 3rd rows) is almost $\pi/2$ so the discovered identity is

$$cos(2x) = sin(\pi/2 - 2x).$$

GP: Symbolic Regression

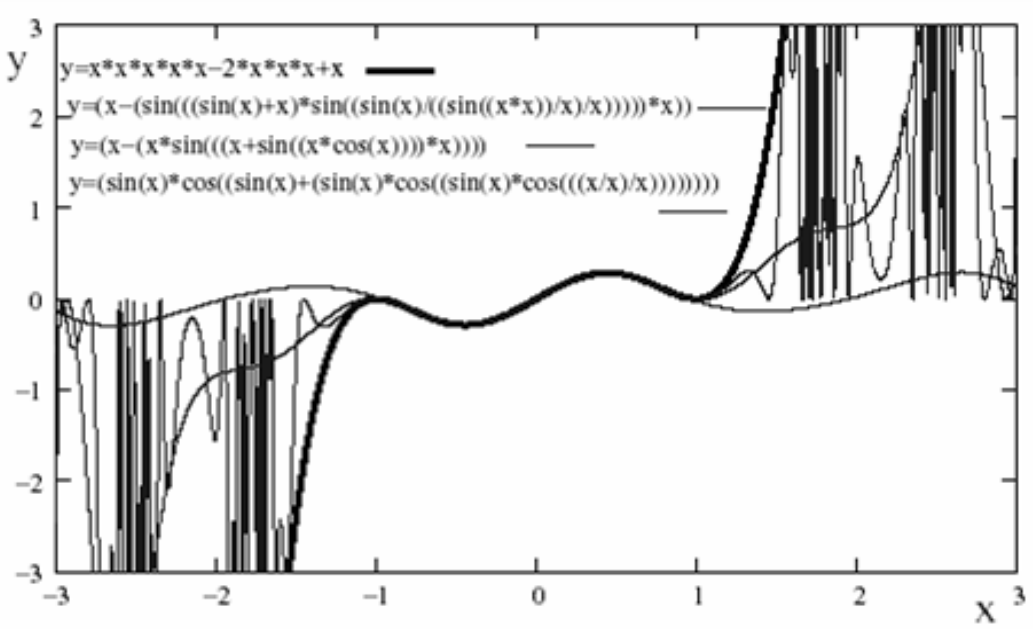
Task is to find a function that fits to training data evenly sampled from interval $< -1.0, 1.0 >$.

GP implementation:

- **Terminal set** $T = \{x\}$.
- **Function set** $F = \{+, -, *, \%, sin, cos\}$.
- **Training cases**: 20 pairs (x_i, y_i), where x_i are values evenly distributed in interval $(-1, 1)$.
- **Fitness**: Sum of errors calculated over all (x_i, y_i) pairs.
- **Stopping criterion**: A solution found that gives the error less than 0.01.

Besides the desired function other three were found

- with a very strange behavior outside the interval of training data,
- though optimal with respect to the defined fitness.



GP: Effect of Proper Primitive Set Selection

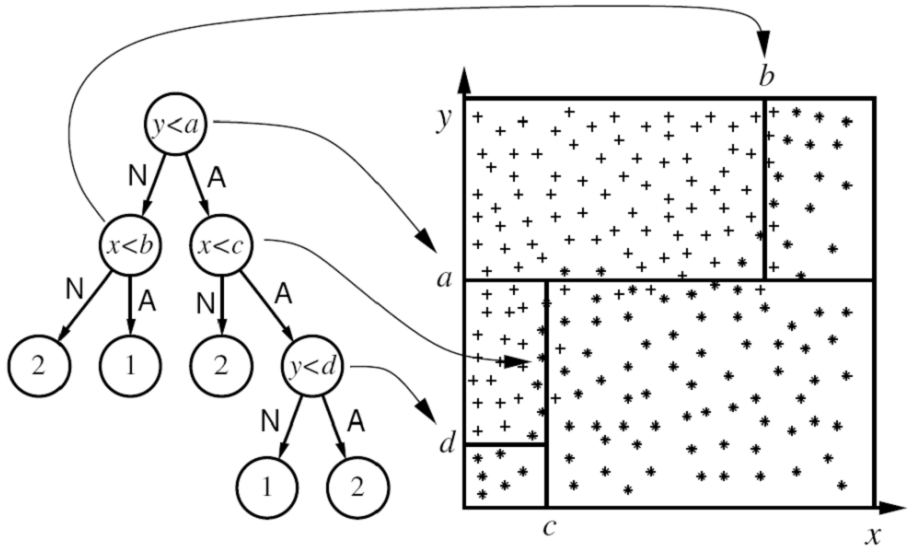
Case study: Decision trees (DT)

- Inner nodes represent simple decisions of type

$$att > const, att = const$$

  $\Rightarrow$ **axis-parallel splits**, in some cases not very efficient way to partition the attribute space.

- Leaf node indicates a class to which the sample, which corresponds to the decisions made along the branch from root to the leaf, belongs.
- Standard learning algorithms – Quinlan's ID3 (Iterative Dichotomiser 3).

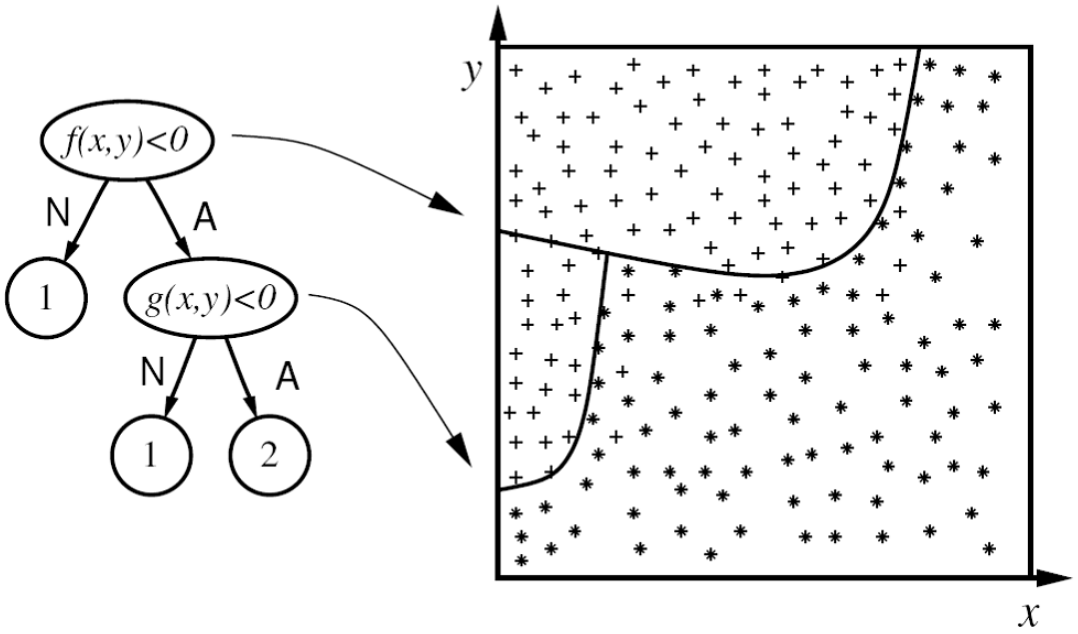


GP: Decision Trees cont.

Oblique/Multivariate DT

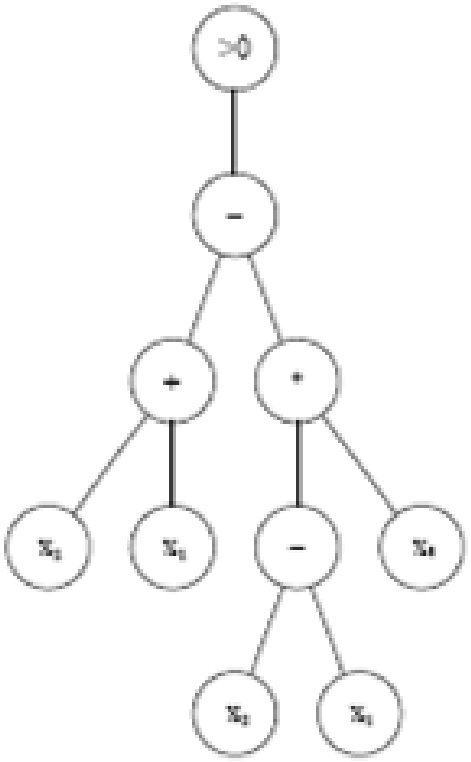
- Inner (decision) nodes represent complex rules (functions).
- More flexible splits are possible.
- But may be hard to understand and interpret.

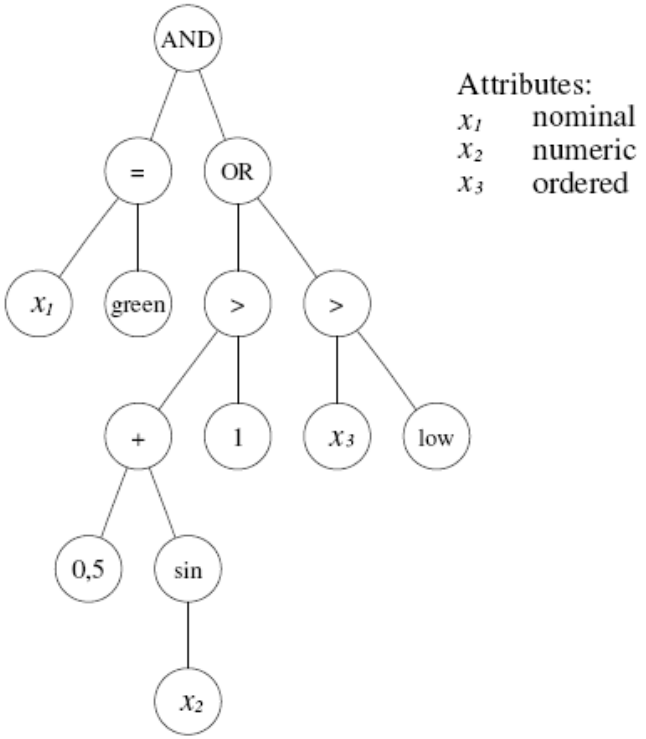
This looks much better, but how to find the rules?



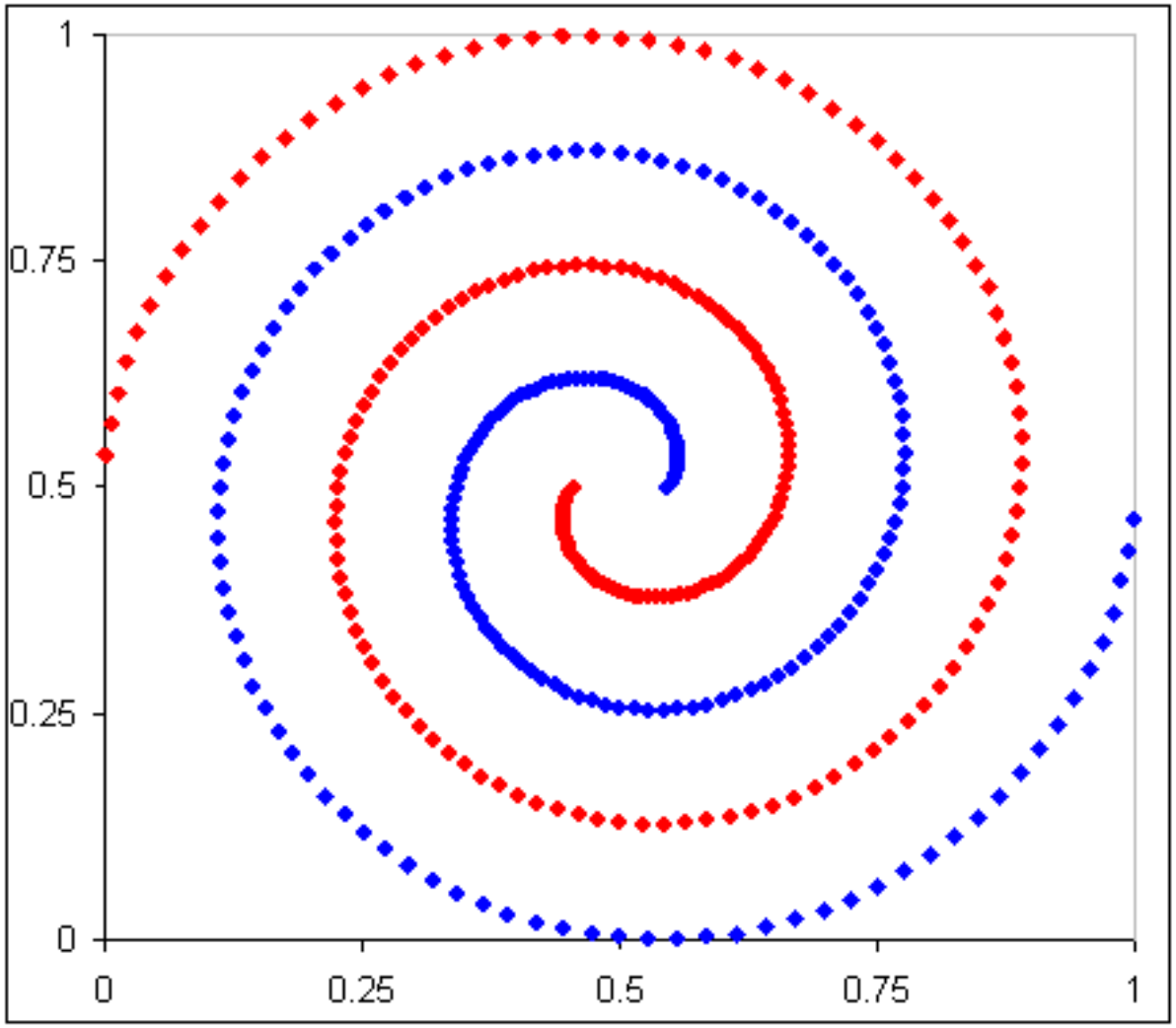
Multivariate DT: Rule Structure

Root node returns boolean value $\{true, false\}$





Intertwined Spirals Problem



Multivariate DT: Effect of Terminals and Functions Selection

Without any prior knowledge A

- Terminals: $T_a = \{x, y, R\}$,
- Functions: $F_a = \{+, -, ?, (> 0)\}$.

With prior knowledge about the circular characteristics of data

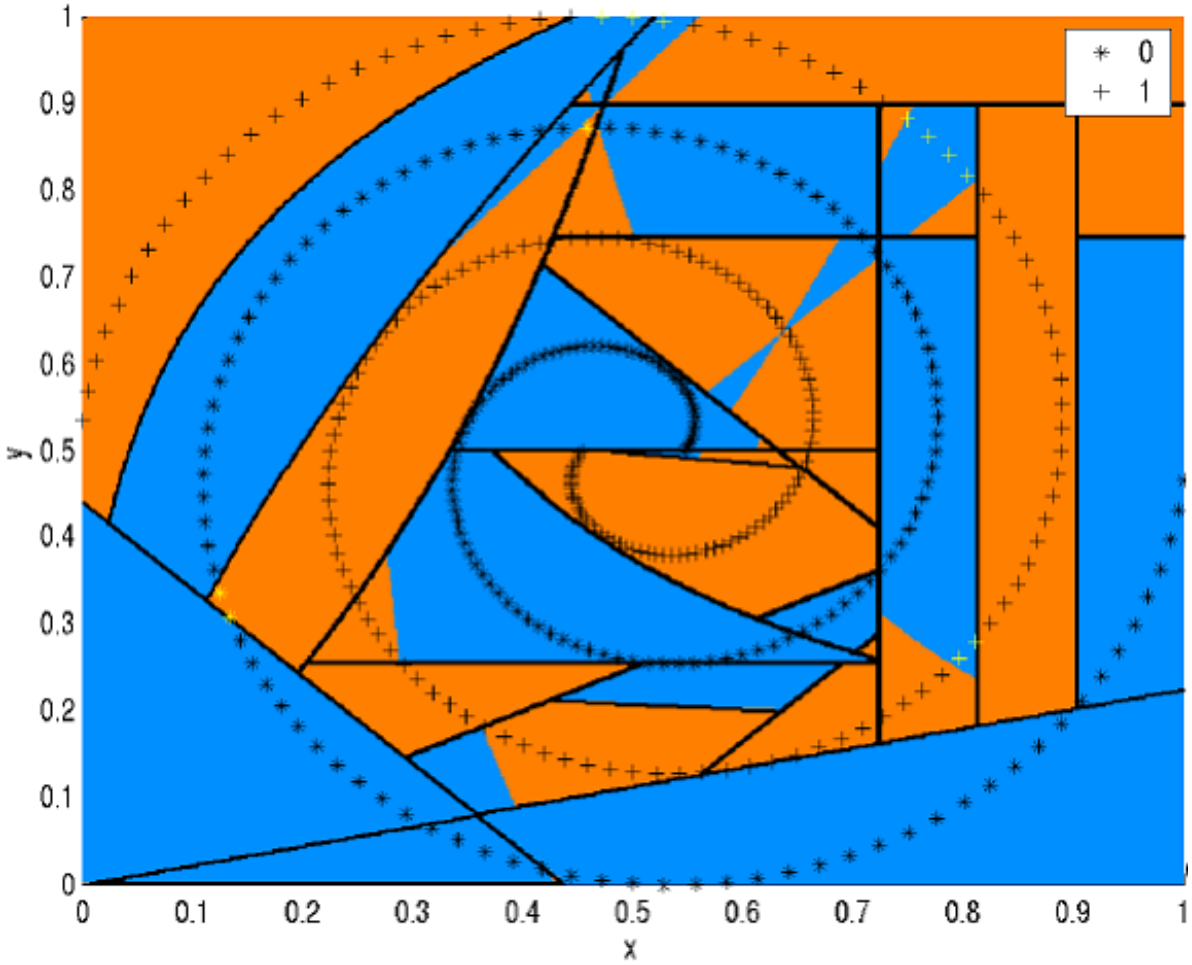
- Polar coordinates.
- Terminals: $T_b = \{\varrho, \varphi, R\}$, where ϱ and φ are radius and phase of data points,

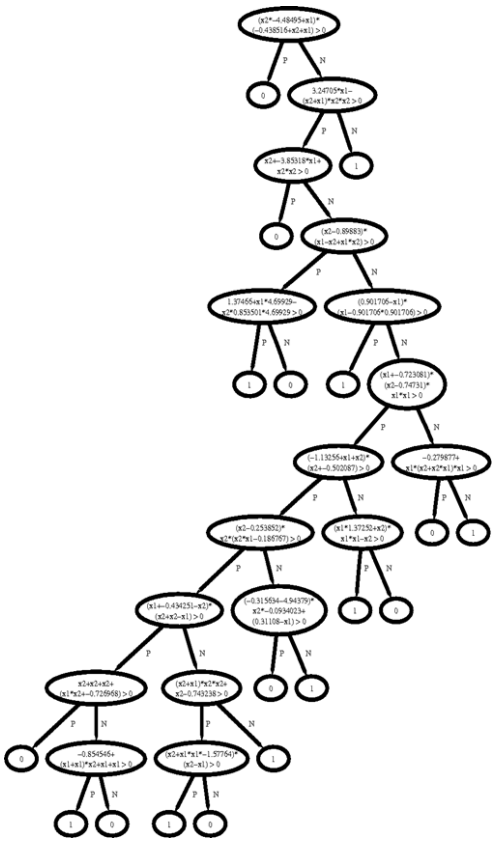
$$\varrho = \sqrt{x^2 + y^2} \text{ and } \varphi = \arctan \frac{y}{x}$$

- Functions: $F_b = \{+, -, *, (> 0), sin\}$.

Multivariate DT: Classifiers Evolved with $\{T_a, F_a\}$

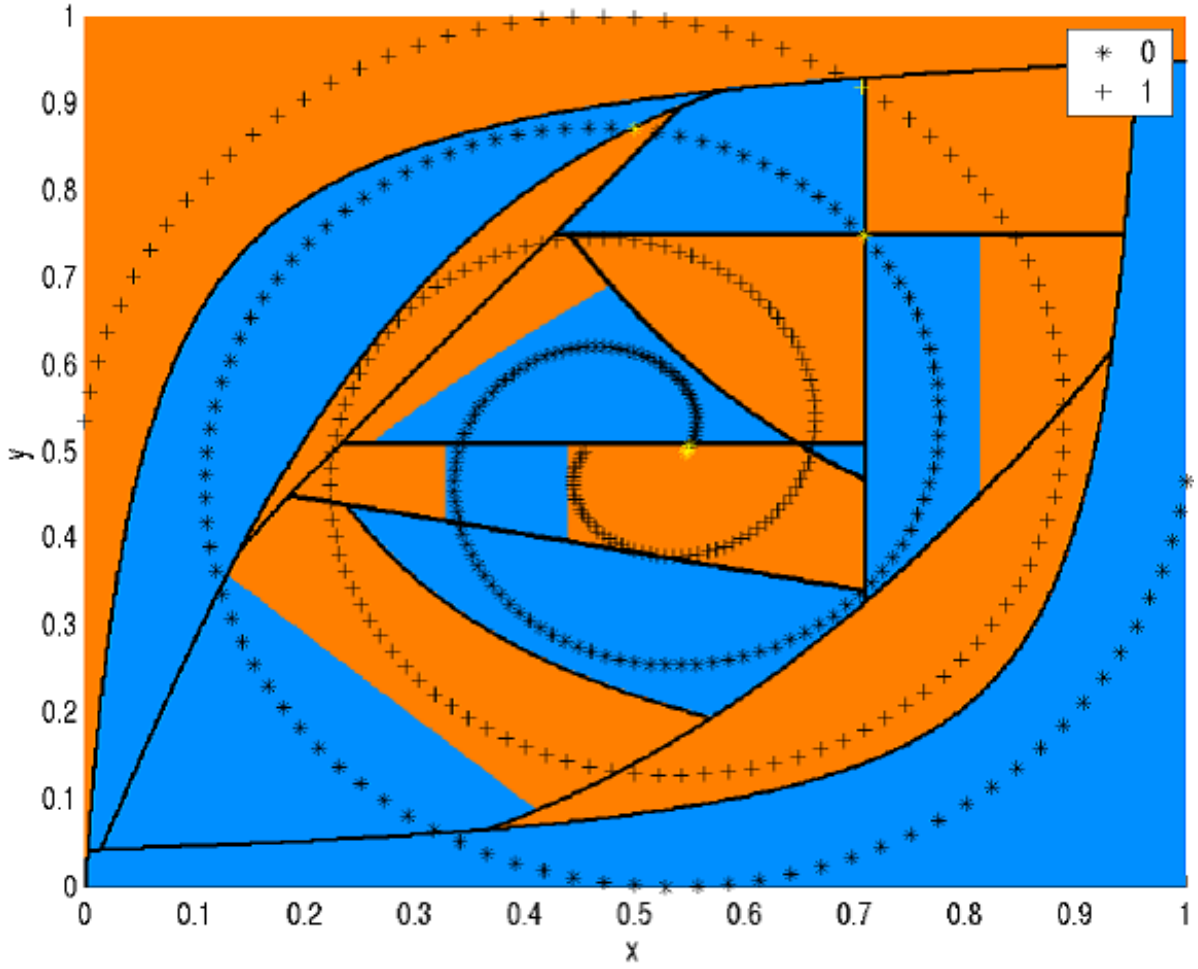
This does not look nice.

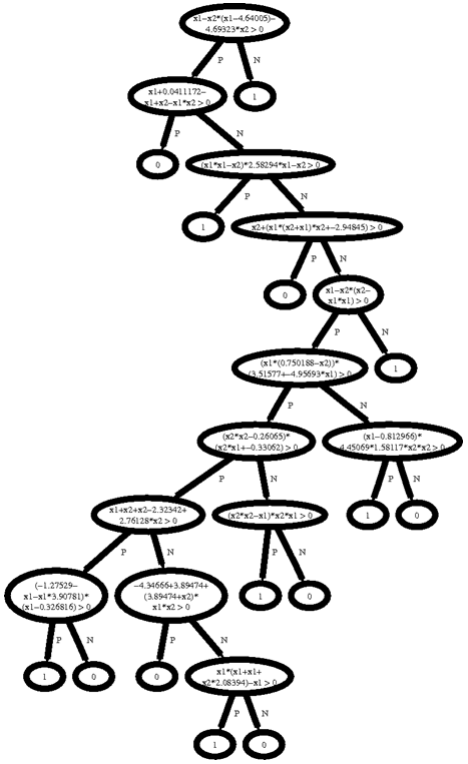




Multivariate DT: Classifiers Evolved with $\{T_a, F_a\}$ cont.

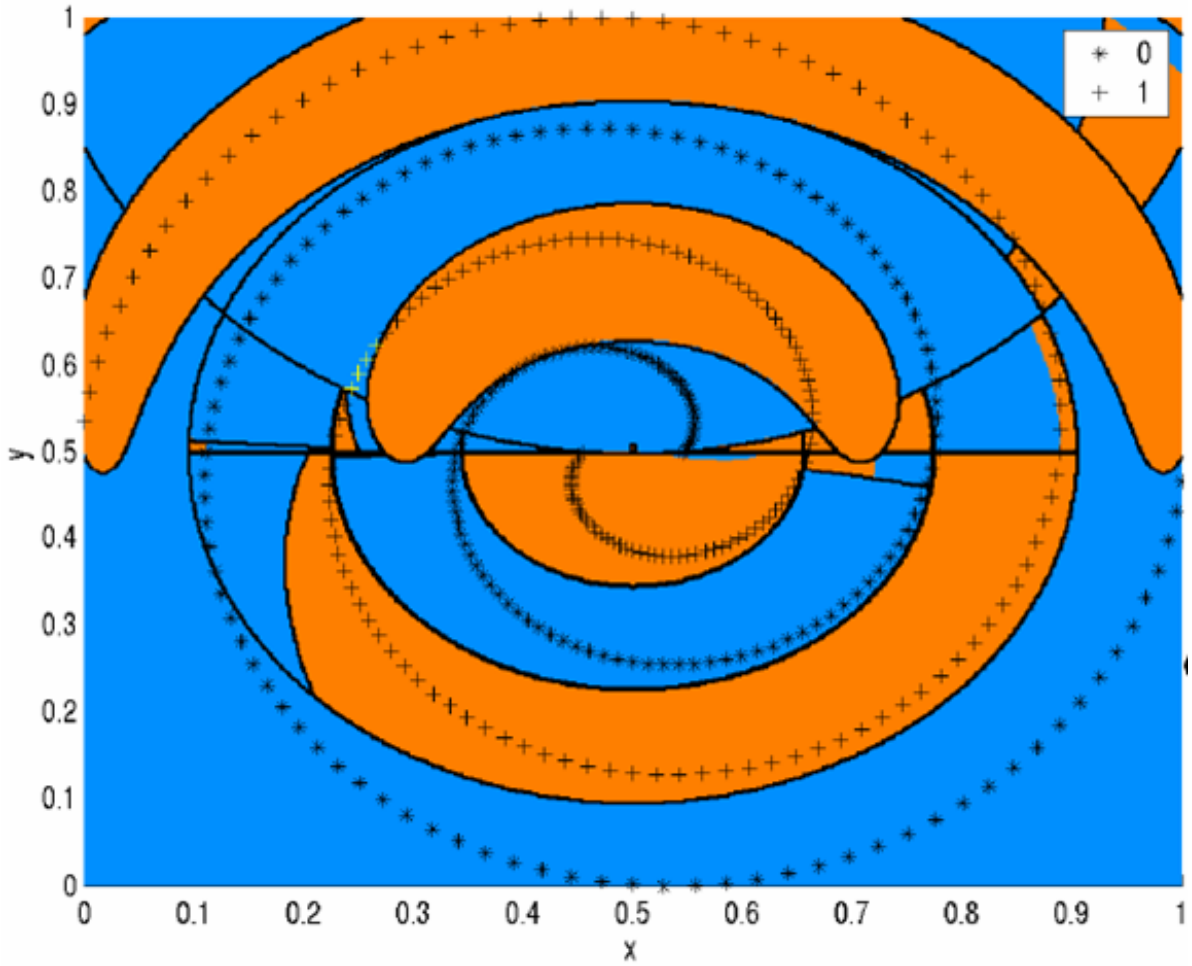
Neither does this one.

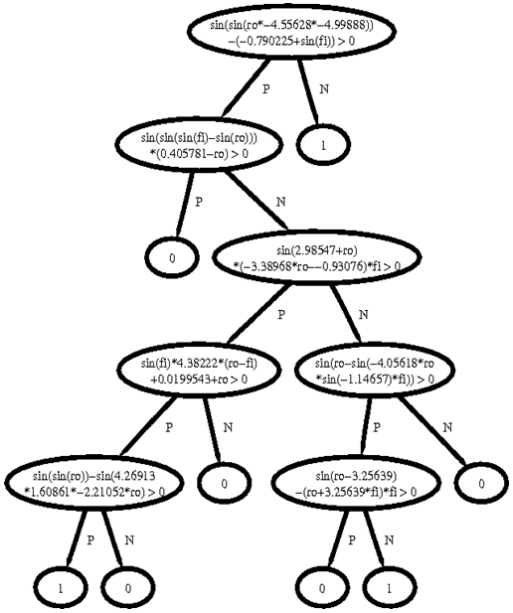




Multivariate DT: Classifiers Evolved with $\{T_b, F_b\}$

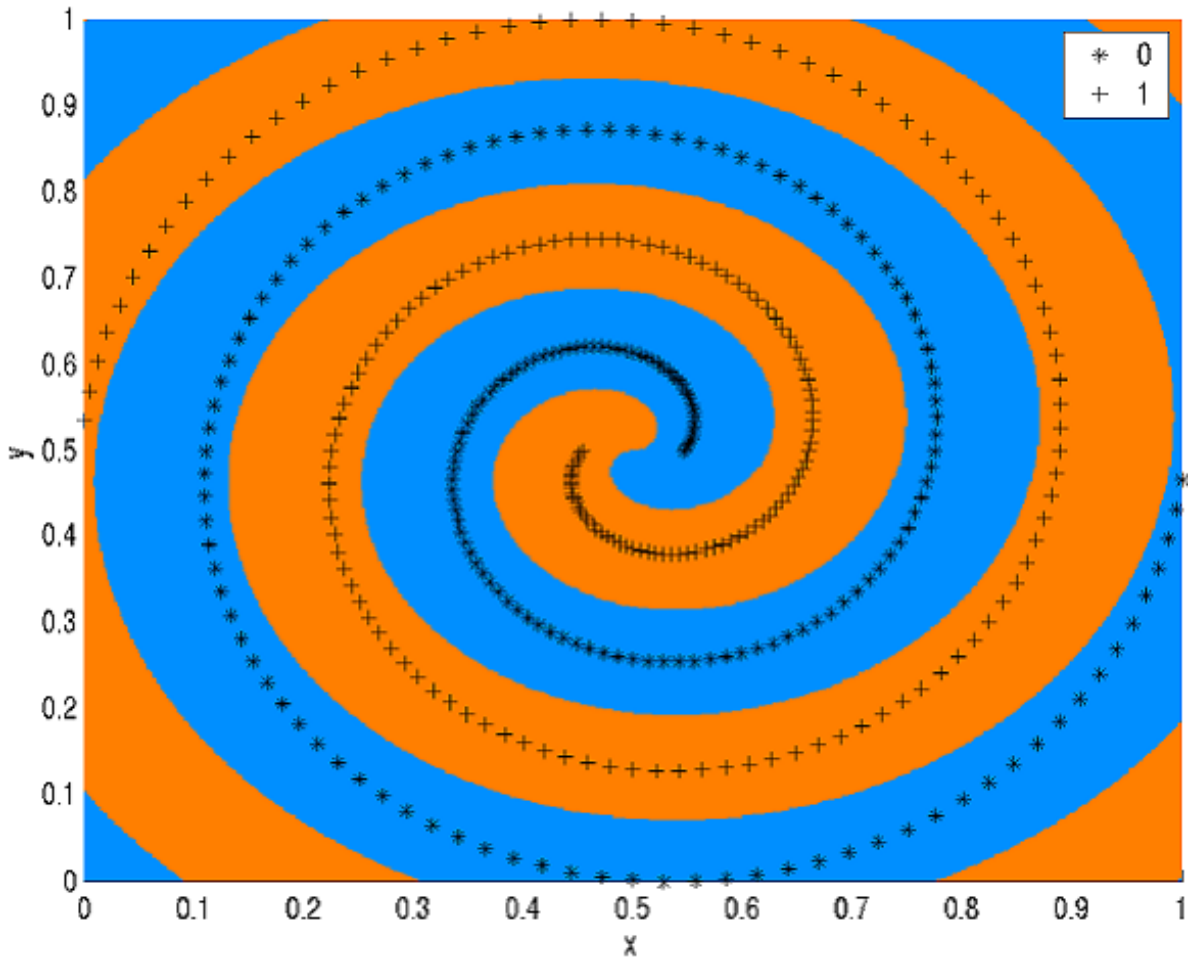
This is better already.

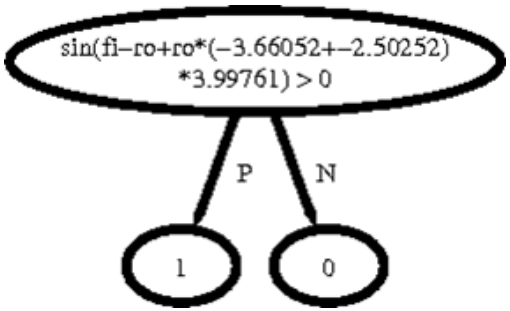




Multivariate DT: Classifiers Evolved with $\{T_b, F_b\}$ cont.

Wow, this one is really nice!!!





Reading

- Poli, R., Langdon, W., McPhee, N.F.: *A Field Guide to Genetic Programming*, 2008, http://www.gp-field-guide.org.uk/