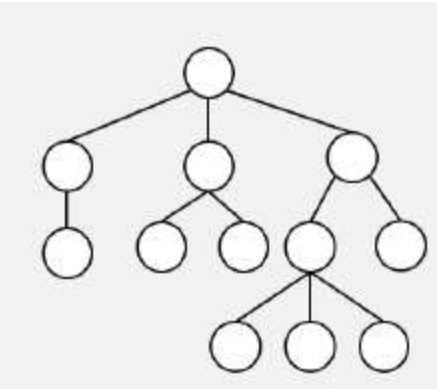
## Genetic Programming (GP)

Genetic programming (GP), proposed by John Koza around the 1990s, is one of the youngest members of the evolutionary algorithm family. It automatically solves problems without requiring the user to know or specify the structure of the solution in advance. The main idea of GP is to evolve a population of computer programs, where individuals are commonly represented as syntax trees, as it is shown in the next figure. While the EAs are typically applied to optimization problems, **GP could instead be positioned in machine learning**.



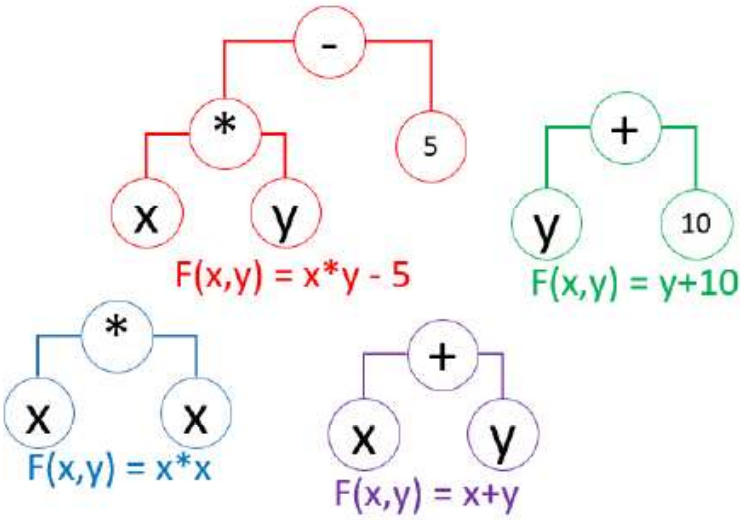
A supervised learning problem consists of mapping several inputs with outputs.

| x | y | F(x,y) |
|---|---|---|
| 5 | 9 | 21 |
| 0 | 7 | 3 |
| 1 | 5 | 5 |
| 4 | 7 | 15 |
| 1 | 2 | 8 |
| 9 | 7 | 75 |
| 5 | 6 | 24 |
| 3 | 8 | 8 |
| 3 | 3 | 13 |
| 2 | 4 | 8 |

$$F(x,y) = ¿?$$

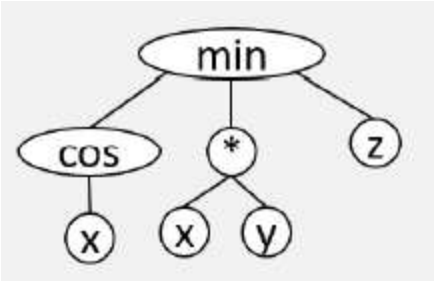
When Genetic Programming is used for solving supervised learning problem, each GP individual represents a solution, as an equation, to solve the problem.

| x | y | F(x,y) |
|---|---|---|
| 5 | 9 | 21 |
| 0 | 7 | 3 |
| 1 | 5 | 5 |
| 4 | 7 | 15 |
| 1 | 2 | 8 |
| 9 | 7 | 75 |
| 5 | 6 | 24 |
| 3 | 8 | 8 |
| 3 | 3 | 13 |
| 2 | 4 | 8 |




## Representation

In Genetic Programming, individuals are usually represented as syntax trees.




Min( cos(x), x*y, z )

The elements of a GP individual are:

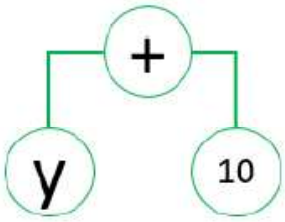
- *Terminals*, which correspond to the leave nodes in a syntax tree. They can be inputs, typically called variables (e.g., x and y), functions without arguments (e.g., rand()), or constants that can be predefined or randomly generated. The terminal set $T$ defines all the terminal elements.

- *Functions*, which correspond to the internal nodes in a syntax tree and can be seen as the operations. The function set $F$ is defined by the problem's nature. For example, for numeric problems, it could be formed by arithmetic operators and trigonometric functions.

## Fitness

A fitness function returns a numeric value that indicates how well the individual solves the problem. Where Genetic Programming is using to solve supervised learning problems, the objective is to find an individual whose function matches inputs with outputs. For each training sample the inputs values can be evaluated in the individual function for calculating the output. In this case, the fitness function must measure how much the individual outputs are similar to the real outputs.

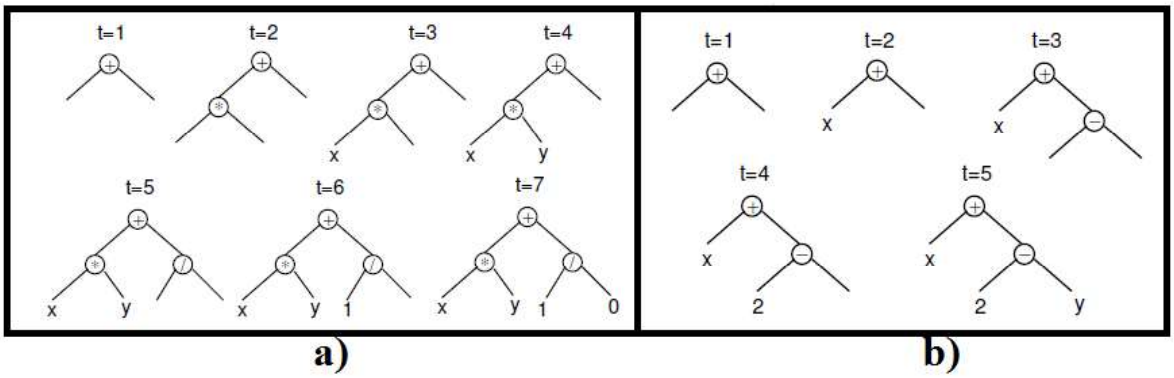
| x | y | F(x,y) |
|---|---|---|
| 5 | 9 | 21 |
| 0 | 7 | 3 |
| 1 | 5 | 5 |
| 4 | 7 | 15 |
| 1 | 2 | 8 |
| 9 | 7 | 75 |
| 5 | 6 | 24 |
| 3 | 8 | 8 |
| 3 | 3 | 13 |
| 2 | 4 | 8 |




| |
|---|
| 19 |
| 17 |
| 15 |
| 17 |
| 12 |
| 17 |
| 16 |
| 18 |
| 13 |
| 14 |

## Initial population

There are three classical techniques to generate the initial population. The first one is called *full*, where all the trees are randomly created, and all the leaves have the same depth. It means, in all levels, except the last one, the nodes are internal nodes selecting elements randomly from the function set $F$ and only in the last level the elements are selected randomly from the terminal set $T$. The second technique is called *grow*; in this case, each node selects an element randomly from either the function set $F$ or the terminal set $T$. Finally, the technique *Ramped half-and-half* where half of the population is created with the *full* technique and the other half with *grow*. After creating the initial population, the fitness of all individuals is calculated. Also, the best program, based on fitness, and its fitness are stored.



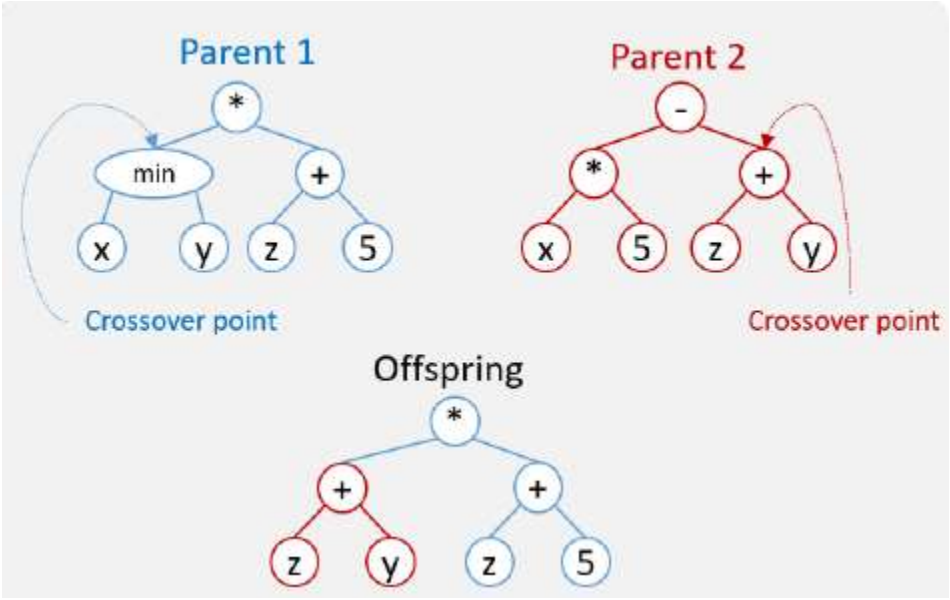

Initial population a) Full b) Grow

## Selection

As most evolutionary algorithms, in GP, the genetic operators (crossover and mutation) are applied to individuals that are stochastically selected based on fitness. This is, more adapted individuals have more chances to be part of the evolutionary process than the ones who are less adapted. In GP, tournament selection is the most popular technique. *Tournament selection* is a direct comparison of fitness among individuals. It can be either binary (two individuals) or with more than two individuals that are randomly selected from the population. The fittest one wins.

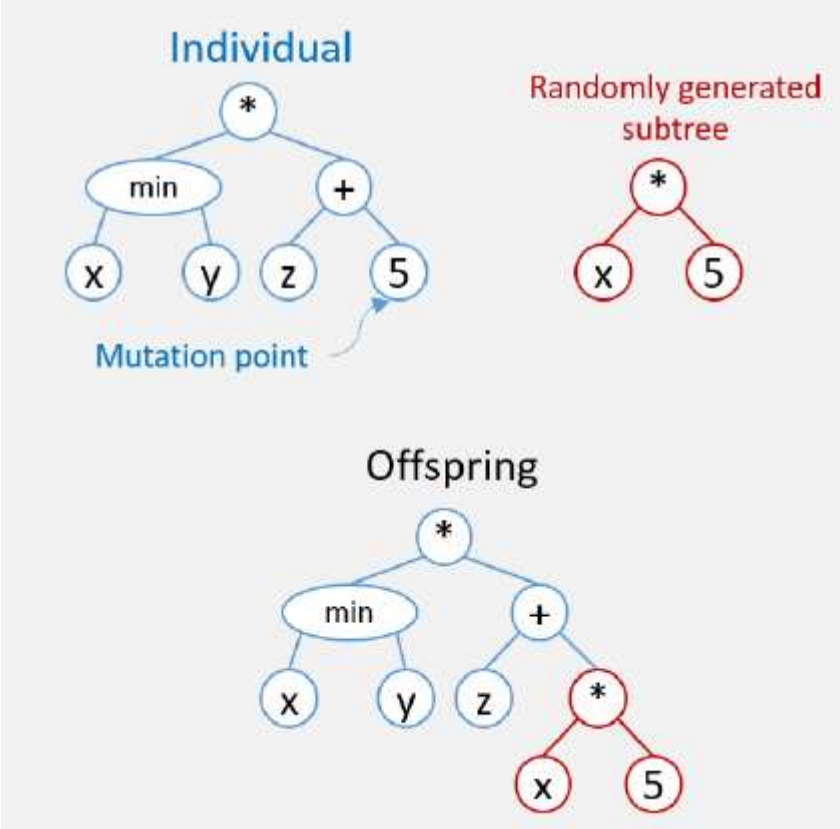
## Crossover

The objective of crossover in GP, as well in other evolutive algorithms, is to combine the characteristics of two or more individuals, called parents, to generate the offspring. The classic crossover consists of randomly selecting a crossover point in each parent and create an offspring based on the first parent but replacing the selected node by the subtree of the second parent on the crossover point.




## Mutation

The goal of mutation is to change an individual. The most used form of mutation in GP, called subtree mutation, randomly selects a mutation point in a tree and substitutes the subtree rooted there with a randomly generated subtree. Another common mutation is point mutation, where a function node is replaced for another one selecting an element from the function set F with the same arity.




## Algorithm

Two different population models exist in Evolutionary Computation: the ***generational model*** and the ***steady-state model***. In the first one, in each generation, we begin with a population of individuals, and several parents are selected from that population to generate the offspring by the application of variation operators. After each generation, the whole population is replaced by its offspring, which is called the next generation. On the other side, in the ***steady-state model***, the entire population is not changed at once, but rather a part of it. Usually, in each iteration, an individual is generated, and it replaces another individual in the population. Most of the recent GP implementation uses the steady-state model. Once the new program is created, it is added to the population, and to maintain the population size, it replaces another one that is chosen using negative selection. In GP, the most used technique for negative selection is negative tournament, it consists of randomly selecting several programs from the population, and the worst of them, based on fitness, is the one that is going to be replaced. In addition, the new program is compared with the best one, and the fittest is kept as the best program.

The population is iteratively changed until a ***termination criterion*** is reached. The termination criterion may be a maximum number of iterations to be run as well as a problem-specific success predicate. Also, a practical criterion is called ***early stopping***, where the training set is divided into a training and a validation sets. The training set is used for guiding the learning process. The evolution stops when the fitness individual on the validation set has not been updated in a defined number of iterations.

The final model corresponds to the fittest individual on the validation set.

**Algorithm 1:** Evolution process of Genetic Programming. The underlined steps correspond to the ones that are analyzed in this dissertation.

Create an initial population $\mathscr{P}$ of programs;
Calculate the fitness of all programs in $\mathscr{P}$;
$best, best_{fitness} \leftarrow$ The best program of $\mathscr{P}$ and its fitness;
**while** *a termination criterion is not reached* **do**
  $parent_1, parent_2 \leftarrow$ Two programs that are selected from $\mathscr{P}$ based on fitness;
  $new \leftarrow$ The program that is the result of applying the variation operators (crossover and mutation) to the programs $parent_1$ and $parent_2$;
  $old \leftarrow$ A program that is chosen from $\mathscr{P}$ using negative selection;
  Remove $old$ from $\mathscr{P}$;
  Add $new$ to $\mathscr{P}$;
  $new_{fitness} \leftarrow$ The fitness of $new$;
  **if** $new_{fitness} > best_{fitness}$ **then**
    $best \leftarrow new$;
    $best_{fitness} \leftarrow new_{fitness}$;
  **end**
**end**
Return $best$;