# Genetic Programming

Rui Mendes

# Advantages of Genetic Programming

- Allows the automatic creation of programs
- Evolves programs that generate solutions instead of the solutions themselves
- More generic
- More powerful

# Disadvantages of Genetic Programming

- Needs more CPU time for training
  - Evaluating programs typically takes longer than evaluating solutions
  - Programs need to be tested in as many situations as possible
- It is often hard to understand the programs that were generated
- Sometimes, it is necessary to use simplifiers in order to be able to understand the result

# Representations

**Trees** A tree represents a program where branches represent non-terminals and leaves represent terminals. The number of sub-trees of a branch is the arity of the function

**Linear** Similar to machine code. Each instruction uses registers for its arguments and storing the result

**Graphs** The program is represented by a graph. Data is usually passed using the stack. The execution flow uses the edges of the graph

# Advantages of Tree Representation

- Intuitive structure for representing programs
- Very similar to Lisp or Syntax Trees
- Very easy to manipulate by genetic operators
- Easy to visualize

# Terminals

- Functions of zero arity
- They represent:
  1. The program inputs
  2. The constants (Random Ephemeral Constants) that are numbers that are generated when GP is initialized and used by all the individuals
  3. Functions with no parameters but that produce side effects

# Functions

| | |
|---|---|
| Booleans | And, Or, Not, Xor |
| Arithmetic | $+$, -, $\times$, $/$ |
| Transcending | $\sin x$, $\cos x$, $\log x$, $e^x$ |
| Attribution | (Assign a 1), (Read a) |
| Conditionals | If Then Else |
| Composing functions | (prog2 fun1 fun2) |
| Others | read sensor, turn left, turn right, move ahead |

# Initialization

**Full** Only non-terminals are used until the maximum depth of the tree is reached. At that time, only terminals are used.

**Grow** When generating a node, both terminals and non-terminals are used. This generates irregular trees

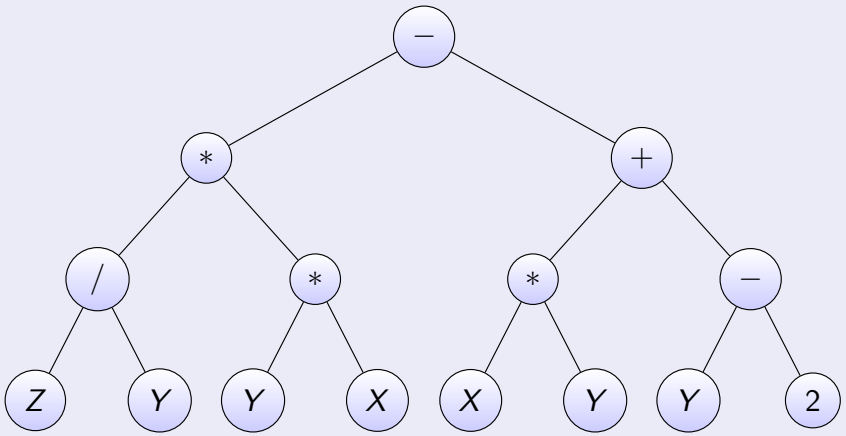
**Ramped half and half** Method that combines **Full** and **Grow**

## Ramped half and half

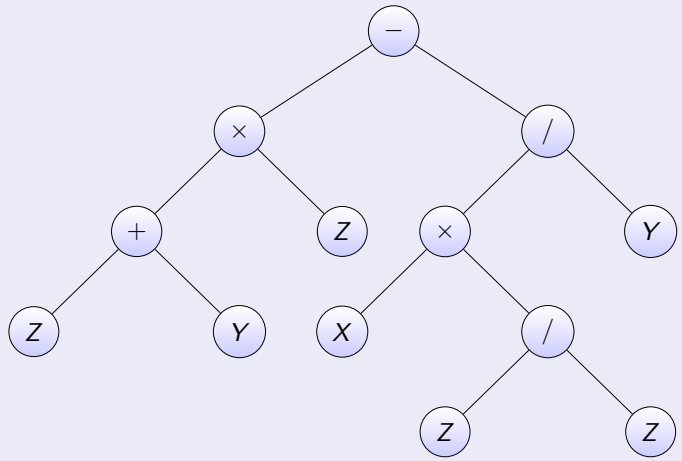
1. The same number of trees is generated for all depths ranging from 2 until the maximum depth
2. For each depth, half of the trees are generated using the *Full* technque while the other half uses the *Grow* technique
3. The same number of trees is generated for all depths ranging from 2 until the maximum depth

# Initialization types

## Full




## Grow

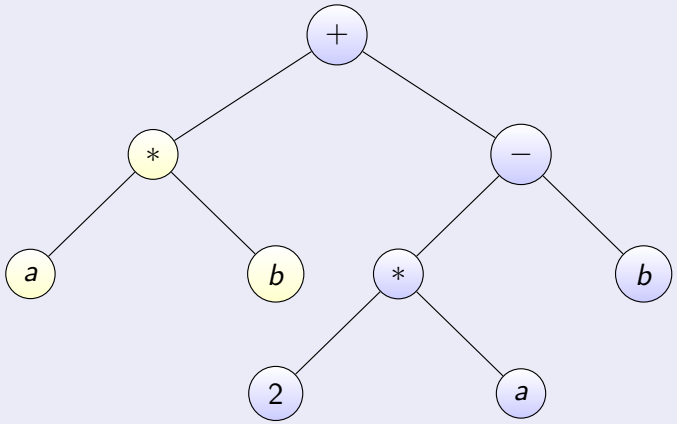


# Crossover

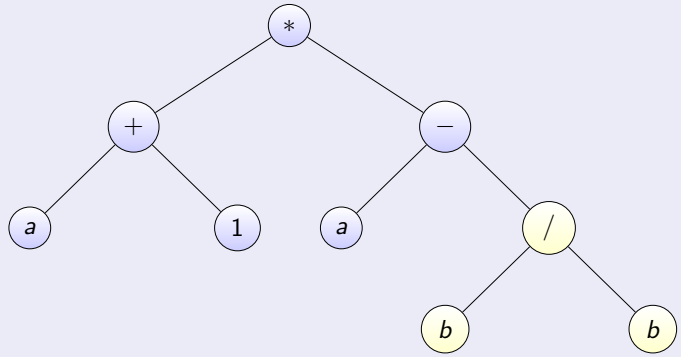
- Two parents are selected for exchanging genetic material
- A subtree is selected at random in each parent
- The subtrees are exchanged between parents
- The exchange is only performed if it doesn't violate the maximum depth constraint
- One of the advantages of crossover between trees is that it can generate different offspring even when both parents have the same information

# Crossover

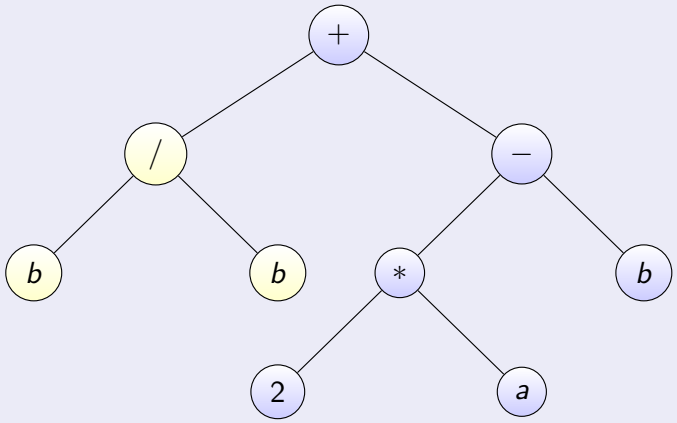
## Parent 1



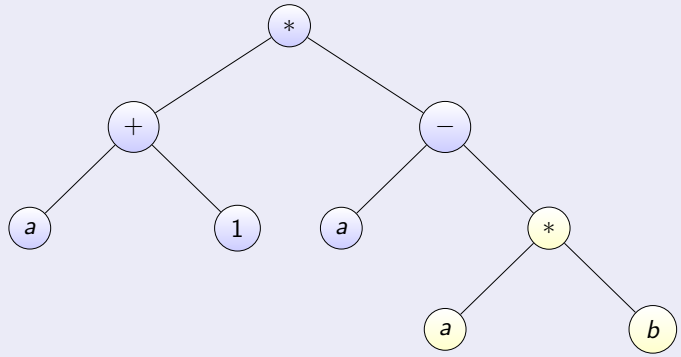

## Parent 2




## Child 1

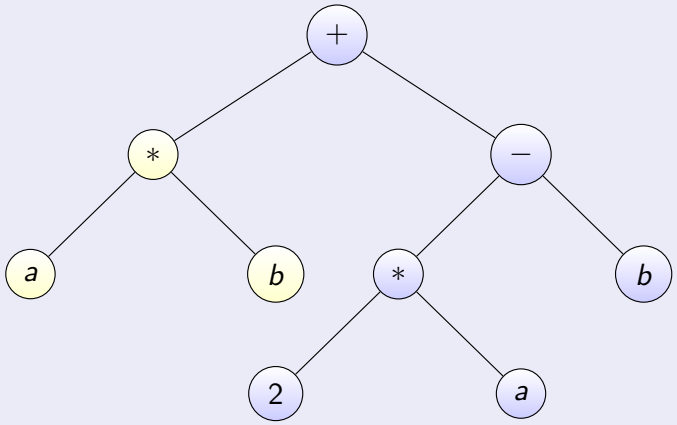



## Child 2

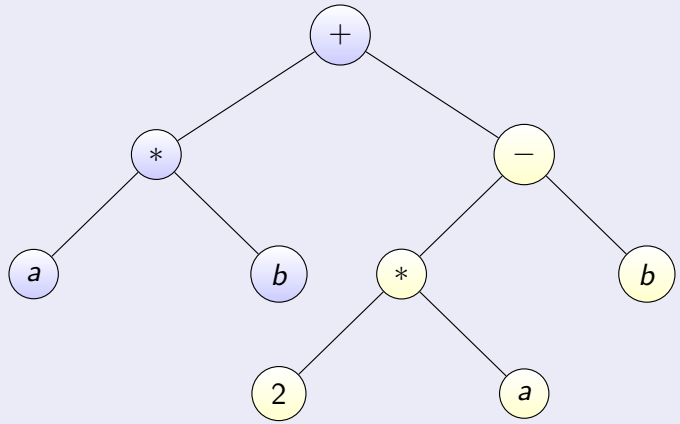


# Crossover between equal individuals

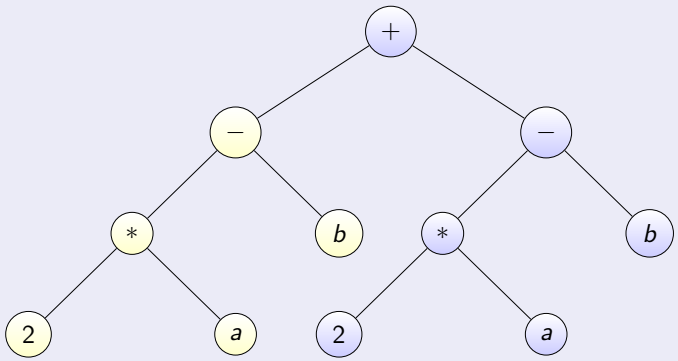
## Parent 1



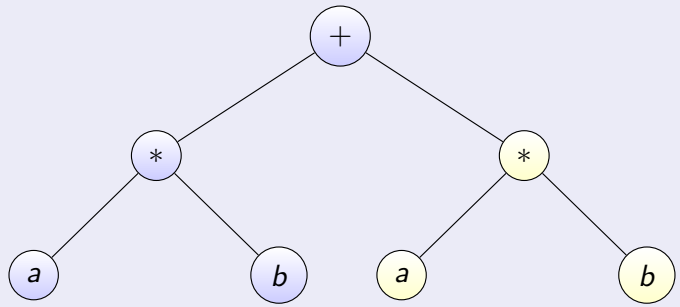

## Parent 2




## Child 1




## Child 2

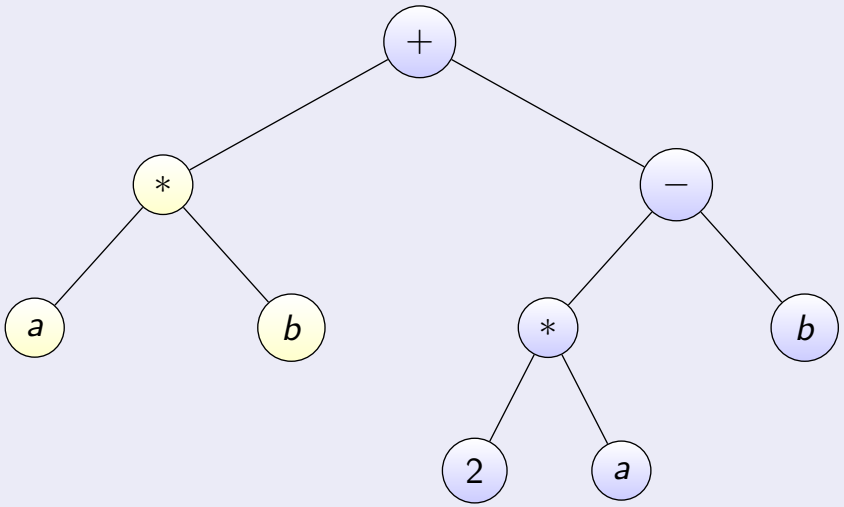


# Mutation

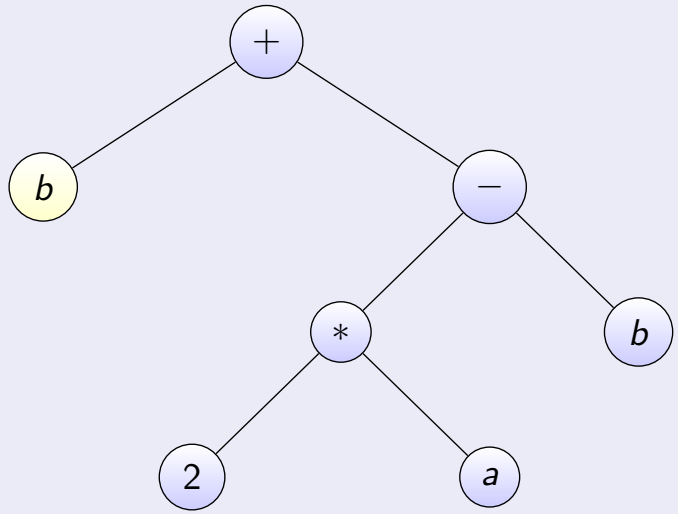
- One individual is chosen to undergo mutation
- A subtree is selected at random
- The subtree is exchanged by a random subtree
- The random subtree is generated while enforcing the maximum depth constraint
- Another possible mutation is to substitute a node by another of the same arity

# Mutation of a subtree

## Before

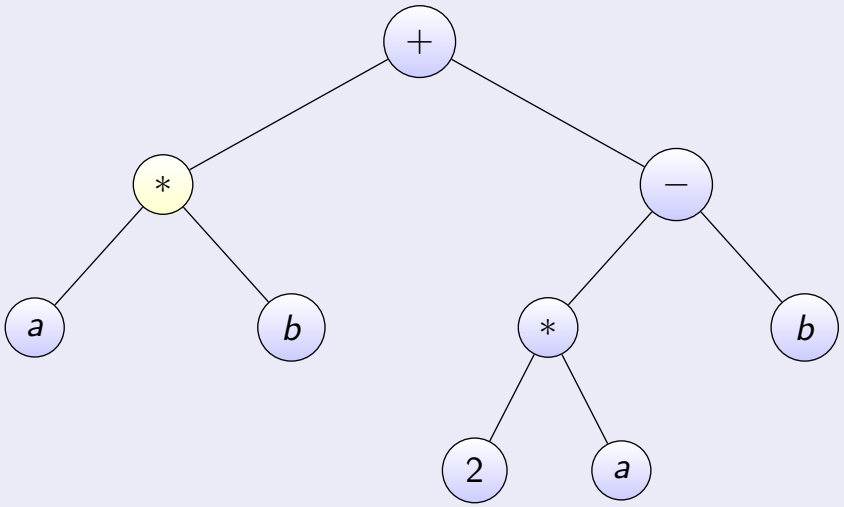



## After

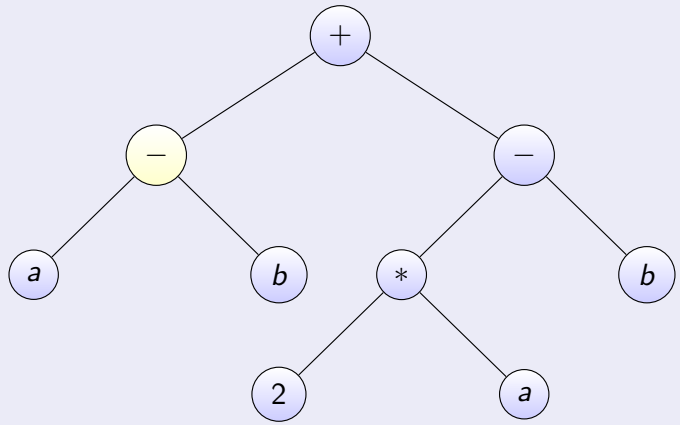


# Mutation of a node

## Before




## After



# Fitness

| | |
|---|---|
| Fitness | A measure of the degree of success of a program |
| Continuity | Small improvements in the program should produce a slightly better fitness; large improvements should translate into a much larger fitness |
| Standardized fitness | Where 0 is given to the optimal solution |
| Normalized fitness | Where the fitness is always between 0 and 1 |

# Examples of Fitness

- The number of collisions of a robot learning how to avoid obstacles
- The average speed of a robot who should learn how to avoid obstacles and always keep in motion
- The error in classification or regression tasks
- The money won by an agent who is learning how to play Poker
- The amount of food gathered by an agent in an artifical life scenario
- The number of goals scored by a robot playing soccer
- The number of hits scored by an agent in a game
- The number of turns an agent survives in a game

# Selection

**Proportional to the fitness** $p_i = \frac{f_i}{\sum_i f_i}$

**$(\mu, \lambda)$ selection** $\lambda$ individuals are generated and the best $\mu$ ones are selected

**Tournament** Several individuals are randomly selected from the population and the best ones create offspring by using the genetic operators and substitute the worse ones

- Two individuals are randomly chosen and the best one creates a mutated copy that substitutes the worse one
- Four individuals are randomly chosen and the best two undergo *crossover* and *mutation* to generate two offspring that substitute the worse ones

# Generational GP

1. Initialize the population
2. Evaluate all the individuals and compute their fitness
3. Until the new population is complete:
    1. Select two parents
    2. Apply genetic operators
    3. Insert offspring into the new population
4. Repeat steps 2 and 3 until reaching the termination criterion
5. Present the best individual in the population

# Steady-State GP

1. Initialize the population
2. Select a random subset of the population to undergo the tournament
3. Rank the individuals
4. Apply the genetic operators to the winners of the tournament
5. Substitute the losers by the offspringg
6. Repeat the previous steps until reaching the stopping criterion
7. Present the best individual in the population

# Advantages of SSGP

- It is not necessary to evaluate all the individuals
- It is only necessary to rank all individuals in the tournament
- This can be done by a competition between the individuals without having a numeric value for the fitness
- It is possible to divide the population in non communicating islands thus parallelizing the algorithm

# Island Model

- The population is divided into several islands
- Each island is standalone
- Some individuals migrate from time to time between islands
- The most usual model uses a circular pattern
- This model often has better results than using the panmictic model

# Parsimony

- Functions used by GP should be sufficiently powerful to encode problem solutions
- Care should be taken in order to avoid using more functions than is strictly necessary
- This increases the problem space and hardens the task of searching for solutions

# Function Closure

- Functions should be able to work with any arguments
- Division must be protected
- This type of protection must be performed for all functions (e.g., square roots or logarithms)

# Parameters

## Configuration Parameters

- Population Size
- Initialization Type
- Genetic Operators
- Operator Probabilities

## Maximum Size

- A big program takes a long time to run
- A big program can encode very specific solutions
- Very specific solutions are hard to generalize
- Maximum number of nodes
- Maximum Depth
- Maximum Number of Instructions

# Applications

- Curve fitting, data modelling
- Image and signal processing
- Time series prediction
- Industrial process control
- Ant foraging in artificial life environments
- Chess endgame players
- Evolve Robocode players
- Heuristics for a Monte Carlo Tree Search Ms Pac-Man agent
- Automatic StarCraft Strategy Generation