Genetic Programming Part 2

Alexandre Bergel
http://bergel.eu
05/12/2018

Outline

1. Recombination and Mutation
2. Subtree crossover
3. Subtree mutation
4. Example

Recombination and Mutation

Essential operations used to produce a better population

Subtree crossover

Given two parents, subtree crossover randomly and independently selects a crossover point (a node) in each parent tree

Then, an offspring is created by

(i) copying the first parent

(ii) selecting a crossover point in that copy

(iii) selecting a subtree in the second parent

(iv) replace the subtree in the copy by a copy of the second subtree

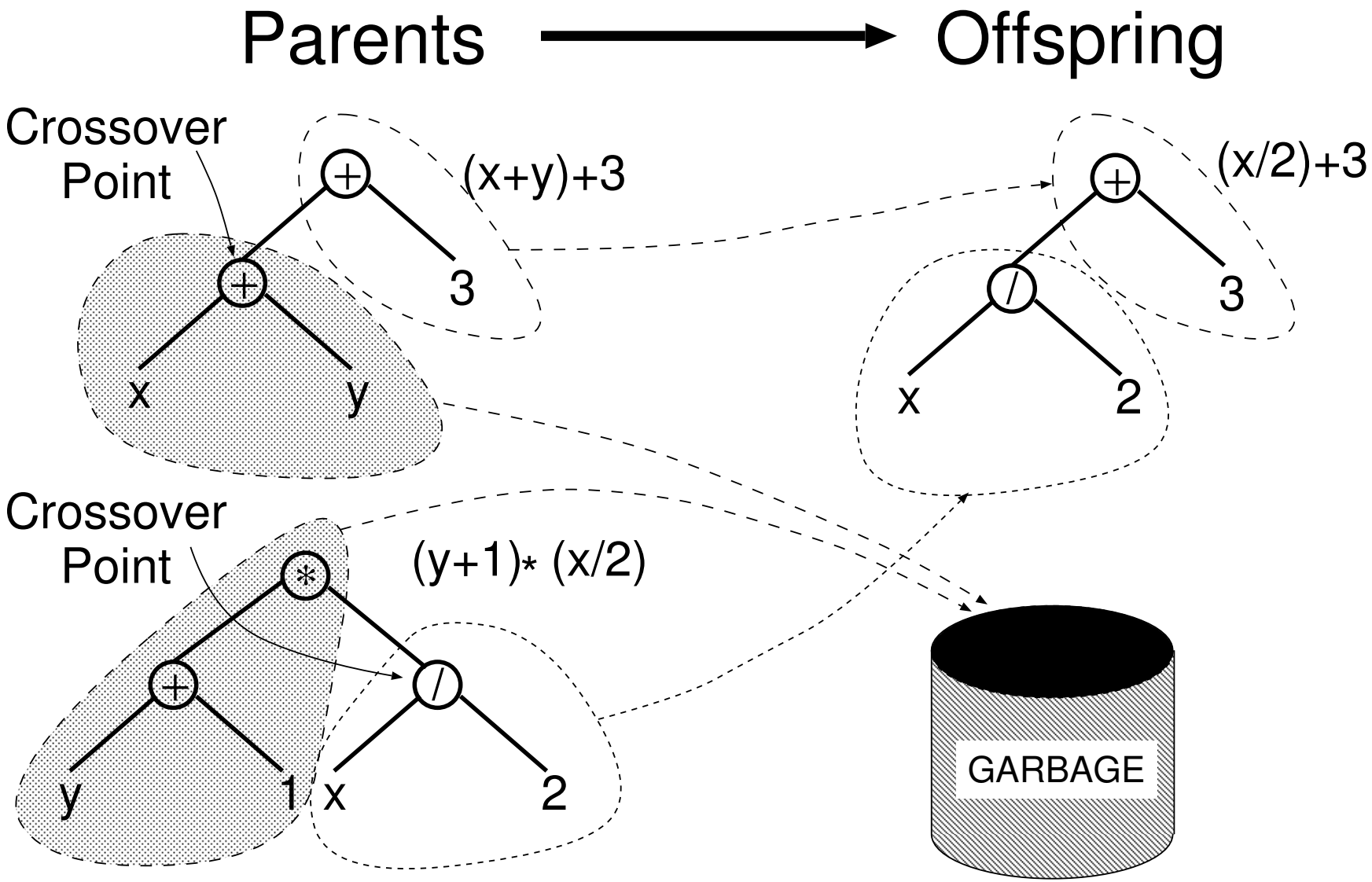
Subtree crossover

Copies are used to avoid disrupting the original individuals

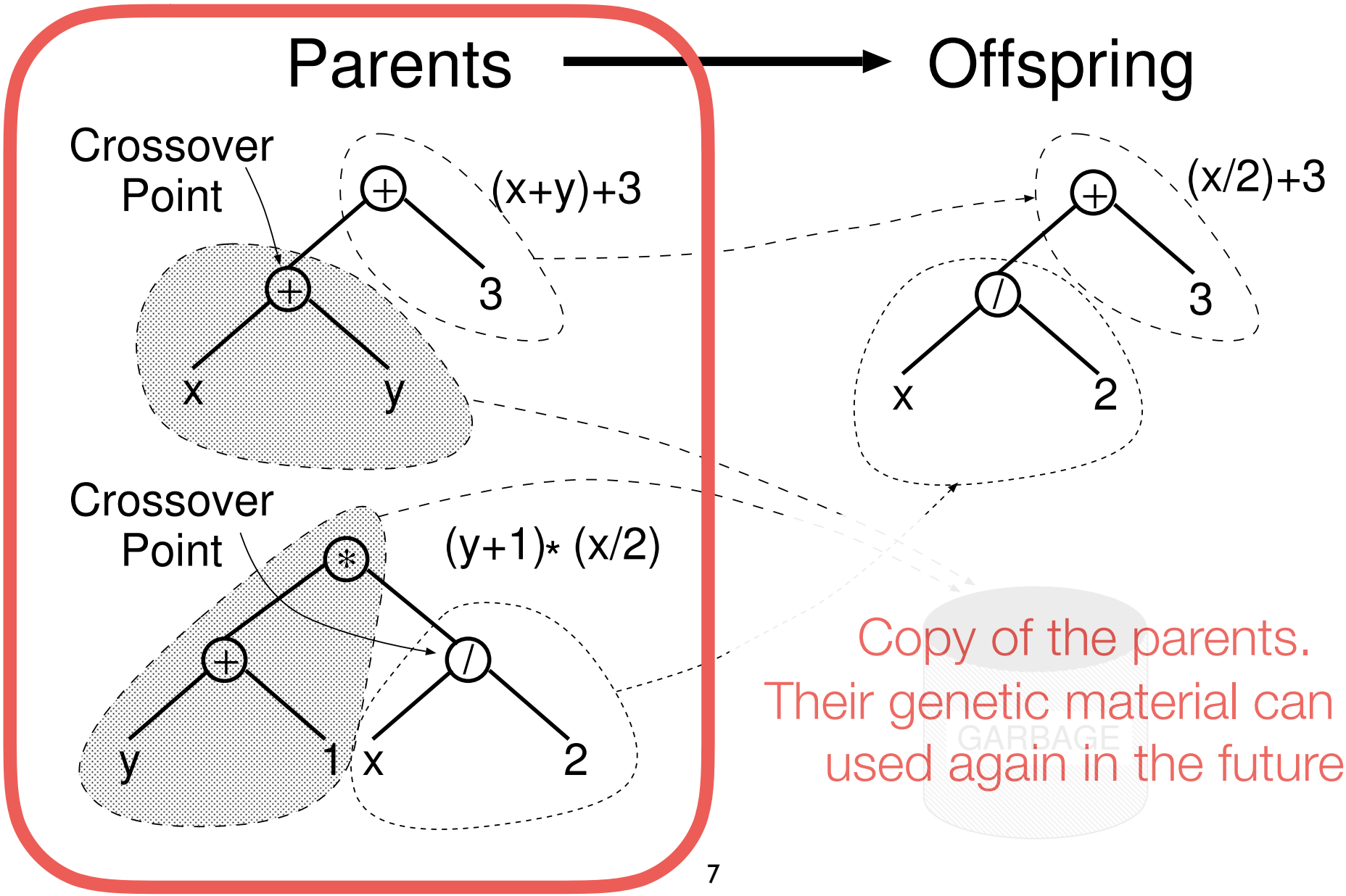
An individual may be multiply selected to be part of the creation of multiple offspring programs

Note that it is also possible to define a version of crossover that return two offsprings, but this is not commonly used

Example


Parents
Offspring
Crossover Point
(x+y)+3
+
+
3
x
y
Crossover Point
(y+1)* (x/2)
*
+
/
y
1
x
2
(x/2)+3
+
/
3
x
2
GARBAGE

Example

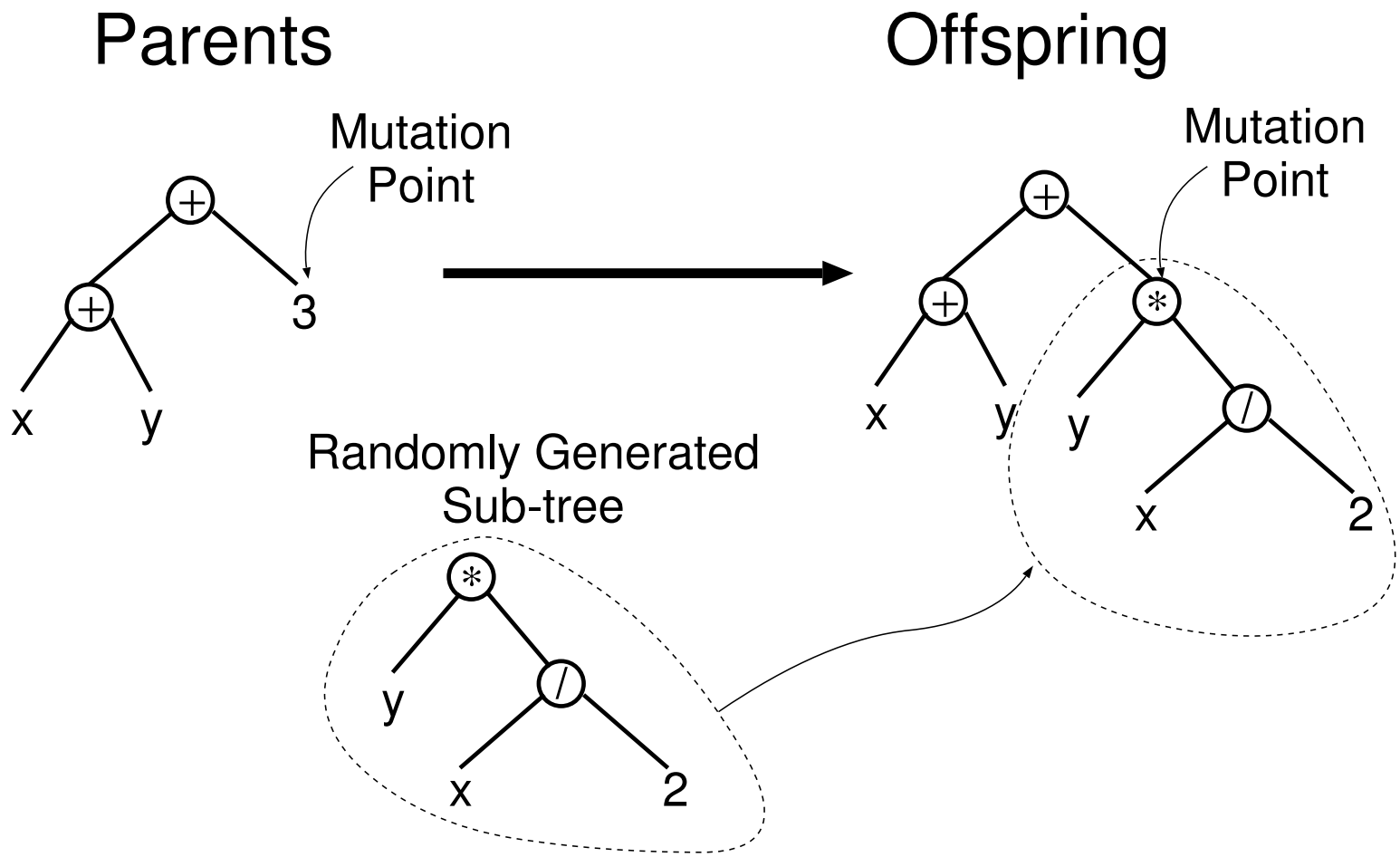

Parents
Offspring
Crossover Point
(x+y)+3
+
+
3
x
y
Crossover Point
(y+1)* (x/2)
*
+
/
y
1
x
2
(x/2)+3
/
3
x
2
Copy of the parents.
Their genetic material can be used again in the future
GARBAGE

Subtree mutation

Randomly selects a mutation point in a tree, and substitutes the subtree with a randomly generated subtree

Subtree mutation may be implemented as a crossover between a program and a newly generated random program

Subtree mutation



Probability of modification

The two modification operations (crossover and mutation) are applied in a probabilistic way

Crossover is applied with the higher probability, 90% or higher

Mutation is much smaller, typically being around 1%

Running Genetic Programming

In order to apply genetic programming to solve a problem, a number of essential steps need to be carefully thought

1. What is the terminal set?
2. What is the function set?
3. What is the fitness measure?
4. What parameters are used to control the execution?
5. What is the termination criterion and what is the result of the run?

Step 1: Terminal set

Genetic Programming (GP) is described as evolving programs

GP is not typically used to evolve programs in the familiar Turing-complete languages used in software development

Instead, it is common to evolve programs (or expressions or formulae) in a more constrained way

Step 1: Terminal set

The terminal set may consist of

Constants, which can be pre-specified, and randomly generated as part of the creation process, or created by mutation

The program's external inputs such as variable names (e.g., x, y)

Function with no arguments, for example, a function dist_to_wall() that returns the distance to an obstacle. Note that functions may do side effects (i.e., changing global state of the program).

Step 1 and Step 2

Domain-specific language are often employed

Step 1 and Step 2 defines such a language

Together, these two steps define the ingredients to create new computer programs.

These two steps will also define the search space GP will explore. This includes all the programs that can be constructed by composing the primitives in all possible ways

Step 2: Function set

The function set is driven by the nature of the problem to solve

In a simple numeric problem, function may be arithmetic functions (+, -, *, /)

All sorts of other functions and constructs may be employed.

Step 3: Fitness function

Steps 1 & 2 defines our search space

Saying which elements or regions of this search space are good is the focus of Step 3

I.e., which regions include programs that solve, or approximately solve, the specified problem

Fitness can be measured in many ways, e.g.,

in terms of the number of errors between the produced output and the desired output

the amount of time (fuel, money, etc) to bring a system to a desired target

Step 3: Fitness function

A fitness function is often represented as a function

f(program) = number

f(program) = 0 means that program is the perfect solution

Or you could also have the opposite, trying to maximize the fitness

Step 4: GP parameters

Many parameters are involved in a GP execution

Population size

Probability of performing the genetic operations

Maximum size for programs

There is no optimal parameter values since this depends very much on the problem to solve

In general, there is no need to spend time on tuning GP to work adequately

Step 4: GP parameters

Having a ramped half-and-half with a depth range of 2 - 6

Traditionally, 90% of children are created by subtree crossover

population size can be 500

The number of generations is limited between 10 and 50

the most productive search is usually performed in early generations

Step 4: GP parameters as a table

Objective:	Find program whose output matches $x^2 + x + 1$ over the range $-1 \leq x \leq +1$.
Function set:	$+$, $-$, % (protected division), and $\times$; all operating on floats
Terminal set:	x, and constants chosen randomly between -5 and $+5$
Fitness:	sum of absolute errors for $x \in \{-1.0, -0.9, \ldots 0.9, 1.0\}$
Selection:	fitness proportionate (roulette wheel) non elitist
Initial pop:	ramped half-and-half (depth 1 to 2. 50% of terminals are constants)
Parameters:	population size 4, 50% subtree crossover, 25% reproduction, 25% subtree mutation, no tree size limits
Termination:	Individual with fitness better than 0.1 found

Step 5: Termination and solution designation

Identifying the termination criterion and the method

Typically, the best-so-far program is designated as the result of the run

Although additional programs and data may be appropriate as well

Example

We want to evolve an expression whose values match those of the quadratic polynomial

p(x) = x^2 + x + 1 in the range [-1, +1]

We want to find the function h, such as h(x) = p(x) for all x in [-1, +1]

Example

Terminal set T = {x, R}, where R = numbers

F = {+, -, *, %}, where % is the protective division

Fitness function = sum of absolute errors measured at different values of x in [-1.0, +1.0]

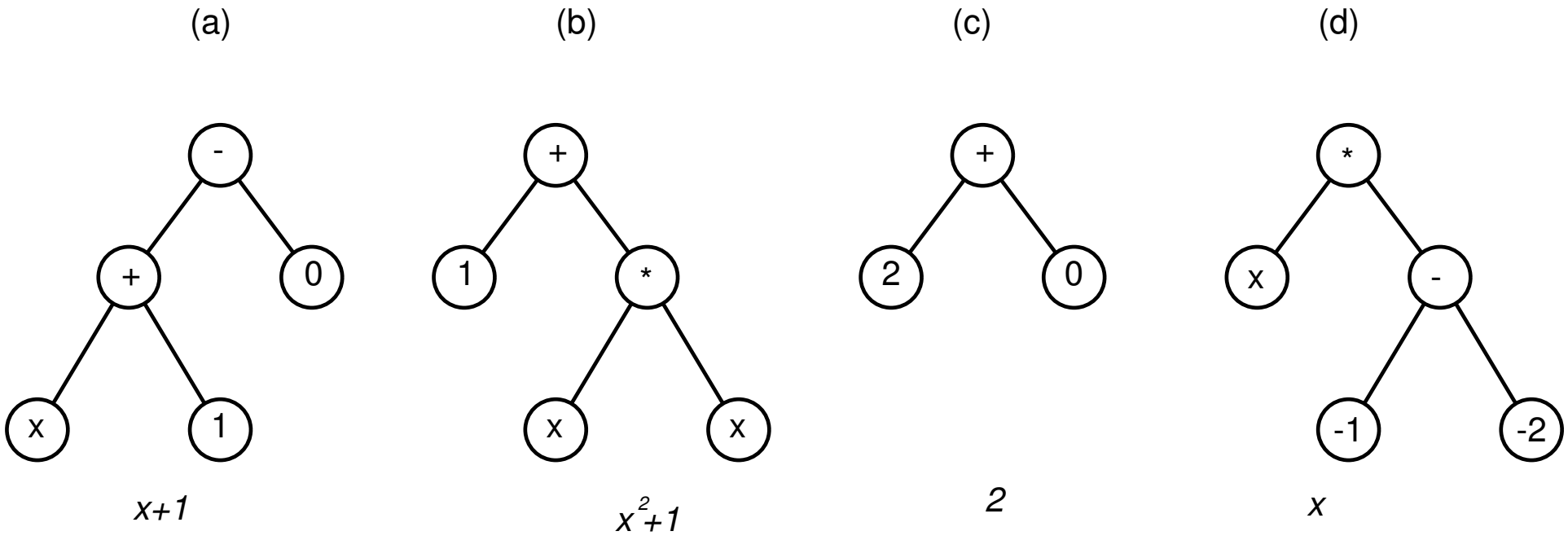
Small fitness is good, while a big fitness is bad

Population size = 4

Cross-over operations = 90%

Mutation = 1%

Example - Initial population



Implementation

Being able to replace a subtree with another subtree

This is necessary to implement the genetic operations

Example

“Des Chiffres et des Lettres” is a popular French TV show.
An equation of the numbers has to match or be close to the expected result
((10 + 9) * (6 + 25)) = 589