Genetic Programming

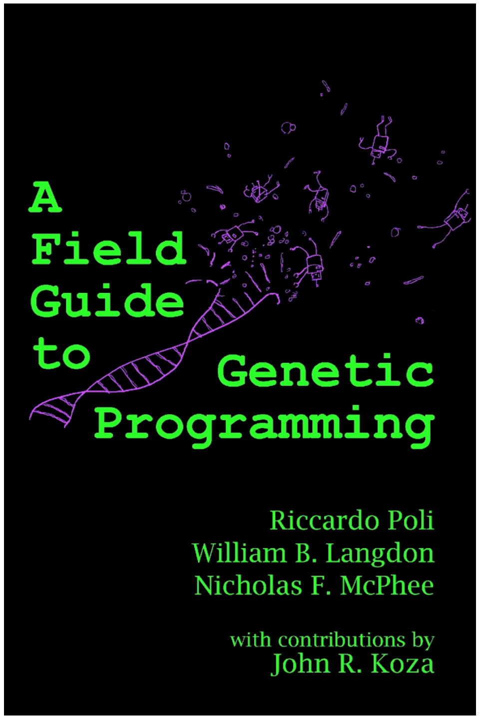
Alexandre Bergel
DCC - University of Chile
http://bergel.eu
30/11/2020

CIENCIAS DE LA COMPUTACIÓN
UNIVERSIDAD DE CHILE

Goal of today

Non trivial improvement of genetic algorithm to cope with tree, called *Genetic Programming*

A bit of lecture



http://www.gp-field-guide.org.uk

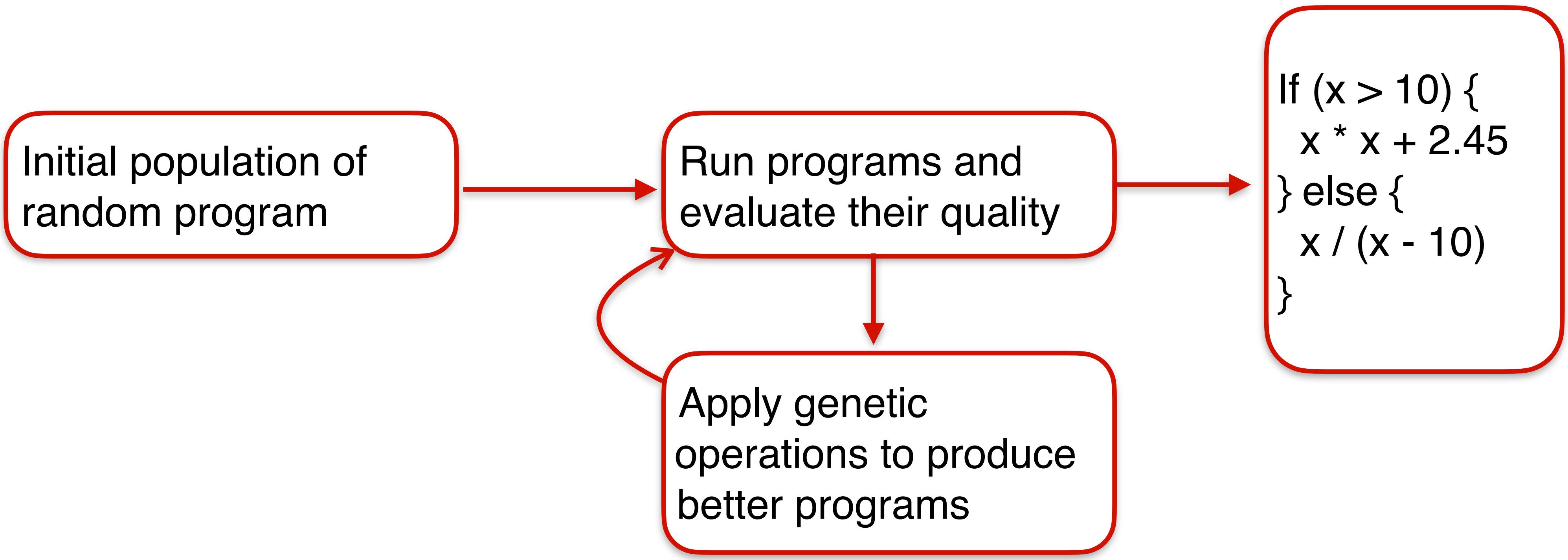
Genetic Programming in a Nutshell

Genetic programming (GP) is about *evolving a population of computer programs*

In GA, each individual represents a linear sequence of genes

In GP, each individual is a *tree* which represents a *computer program*

Genetic Programming in a Nutshell



Genetic Programming in a Nutshell

In principle, genetic programming is very close to genetic algorithm

Fitness function takes as argument a *computer program* and returns *a numerical value*

Advanced GA techniques *equally apply* to GP (e.g., elitism, multi-objective)

Genetic Programming in a Nutshell

GP reasons about *computer programs*

GP takes *programs as input*, and produces better programs

A program is meant *to solve a particular problem*

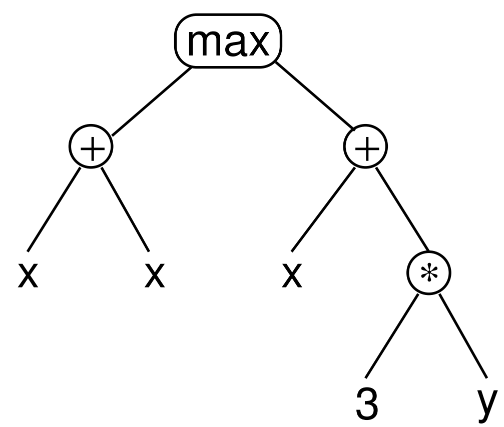
Program

In GP, programs are usually expressed as Abstract Syntax Tree (AST)

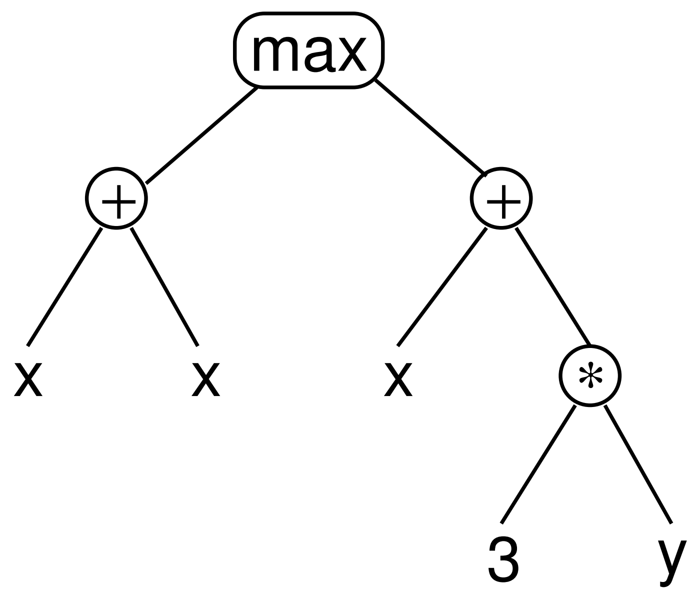
For example, consider the following short program

```
max(x + x, x + 3 * y)
```

This program can be represented as a tree:



Program as a tree



Each element of the tree is a *node*

Variables and *constants* (x, y, 3) are *leaves* of the tree

Internal nodes are called *functions*

functions + terminals = *primitive set* of a GP system

Program

The notion of program in GP is *more general* to what a Java or Python programmer will consider

Program in GP are often *not* written with a general purpose language (e.g., C, Java)

Specific languages are used instead

These languages are often very simple and are *designed to solve a particular problem*

CIENCIAS DE LA COMPUTACIÓN
UNIVERSIDAD DE CHILE

Abstract Syntax Tree

An *Abstract Syntax Tree* is a representation of a source code text, which is easy for the computer to manipulate

Note that the AST representation is essential to many computing aspects:

- *compilation*: transforming AST intro virtual machine byte codes
- *refactoring engine*: transforming an AST into another AST
- *quality rule engine*: computing metrics over AST

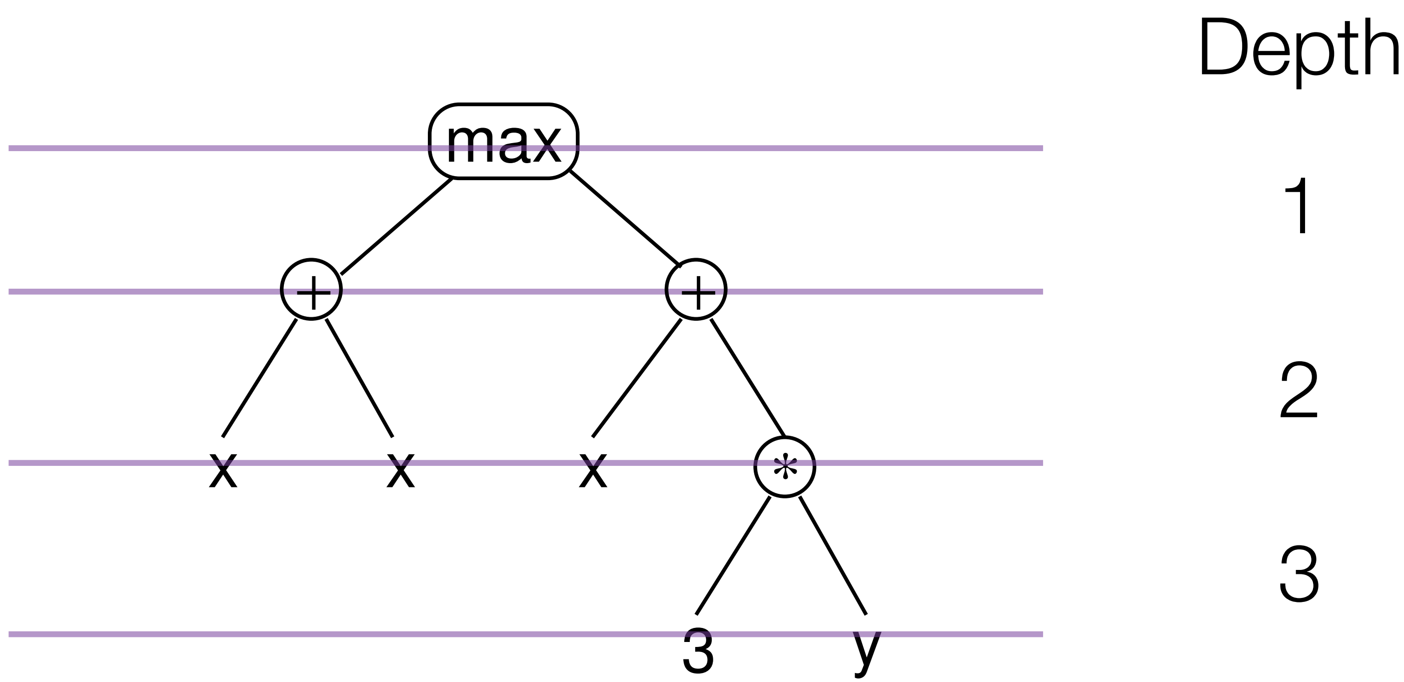
Initializing the population

The initial population is randomly generated (as in GA)

The initial individuals are generated so that they do *not exceed a user specified maximum depth*

The *depth of a node* is the number of edges that need to be traversed to reach the root node

Initializing the population



Genetic operations

Crossover: the creation of a child program by combining randomly chosen parts from two selected parent programs

Mutation: The creation of a new child program by randomly altering a randomly chosen part of a selected parent program

Genetic operations

In GA, genetic operations are designed to operate on a *linear sequence* of genes

This is a relatively simple setting in which

- All the individuals *have the same length*
- Crossover and mutation *operations are very simple*

Trees

In GP, each individual is a tree

Each individual has its own tree shape

- Depth of the tree
- Number of functions and terminal nodes

Essential operations on trees

To adequately performs the genetic operations and evaluate the fitness function, it is essential to

Copy: tree must be duplicated when needed. A genetic operation must produce a new individual, and must not modify the existing individuals

Print: a tree must be printable. This is necessary to see the result

Evaluate: The fitness function is a function that receives a program (i.e., a tree) as argument. A tree must be evaluated therefore

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

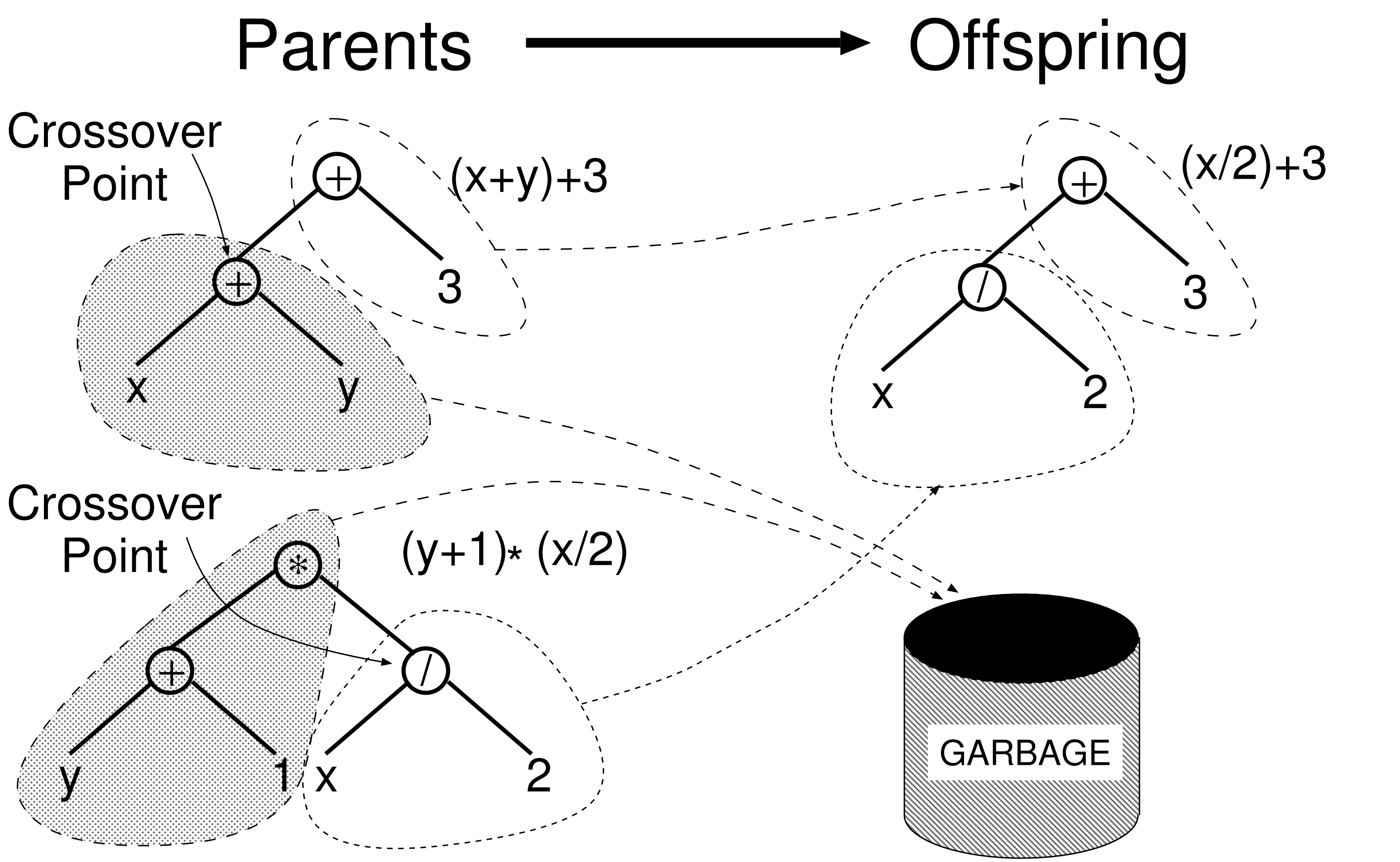
Subtree crossover

Copies are used to *avoid affecting the original* individuals

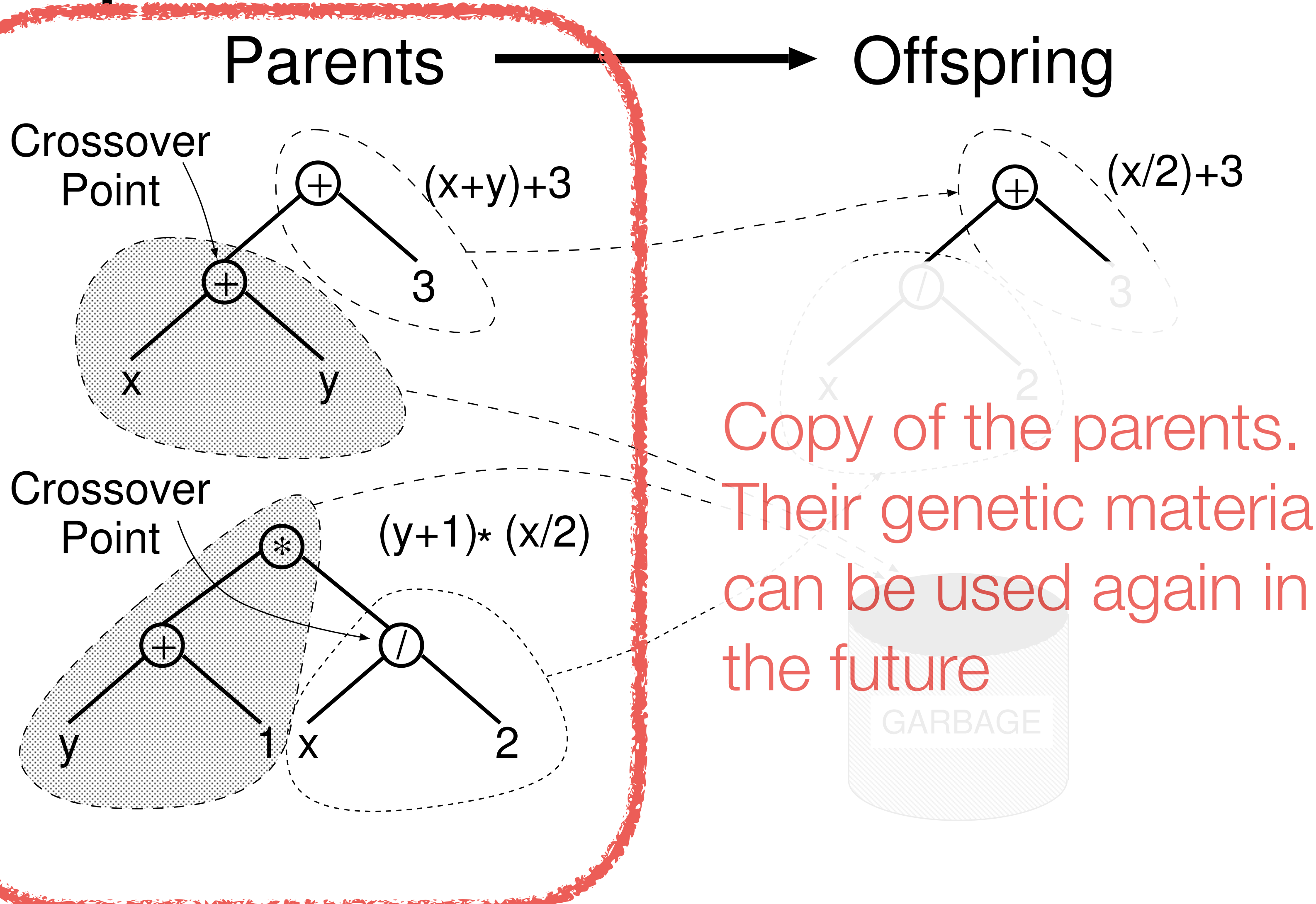
An individual may be multiply selected to be part of the creation of multiple offspring programs

Note that it is also possible to define a version of *crossover that returns two offsprings* (as in GA)

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



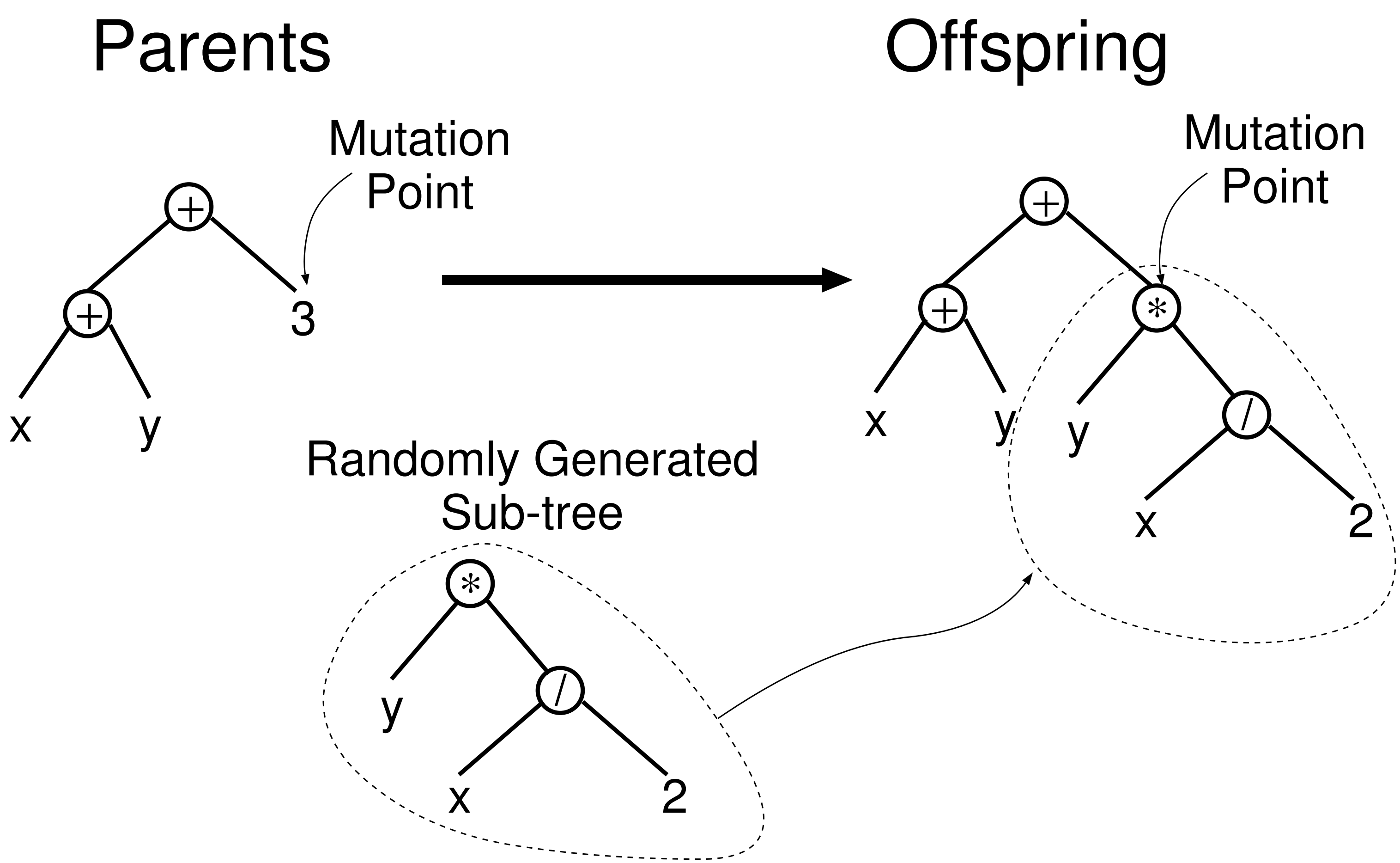
Copy of the parents. Their genetic material can be used again in the future

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



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 about *making programs evolve*

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

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: define the 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 resources* (e.g., time, fuel, money) to bring a system to a desired target

Step 4: GP parameters

Many parameters are involved in a GP execution

- Population size
- Probability of performing the genetic operations
- Maximum size for programs (width and depth)

There is *no general 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

Commonly, 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*

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

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

Conclusion

GP has many applications

- Software program repair
- Defining AI for video games
- Solving numerical problems
- Software patches

dcc

CIENCIAS DE LA COMPUTACIÓN
UNIVERSIDAD DE CHILE

www.dcc.uchile.cl

/ DCCUCHILE