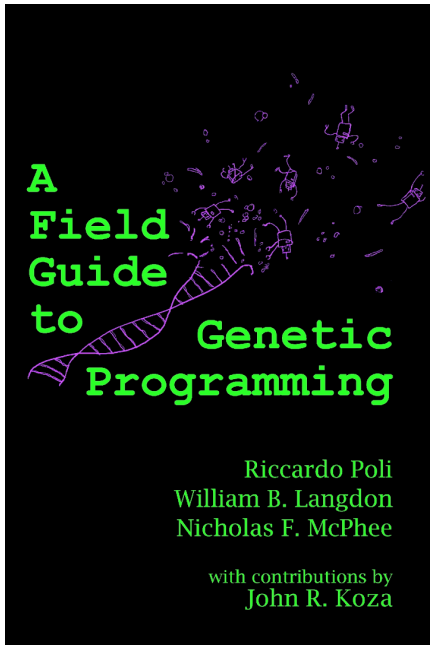
Genetic Programming
Part 1

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

Source



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

Outline

1. In a Nutshell
2. Program
3. Population
4. Selection
5. Implementation

Genetic Programming in a Nutshell

Evolutionary computation technique that automatically solves problems without specifying the form or structure of the solution in advance

GP is a *systematic*, *domain-independent* method for getting computers to solve problems automatically starting from a high-level statement of what needs to be done

In a Nutshell

Genetic programming is about evolving a population of computer programs

Generation by generation

A new generation is, hopefully, better than the previous generation

In a Nutshell

stochastic (stə'kastik), adjective: randomly determined; having a random probability distribution or pattern that may be analyzed statistically but may not be predicted precisely.

GP *stochastically* transforms populations of programs into new, hopefully better, population of programs

Like nature, GP is a random process and it can never guarantee results

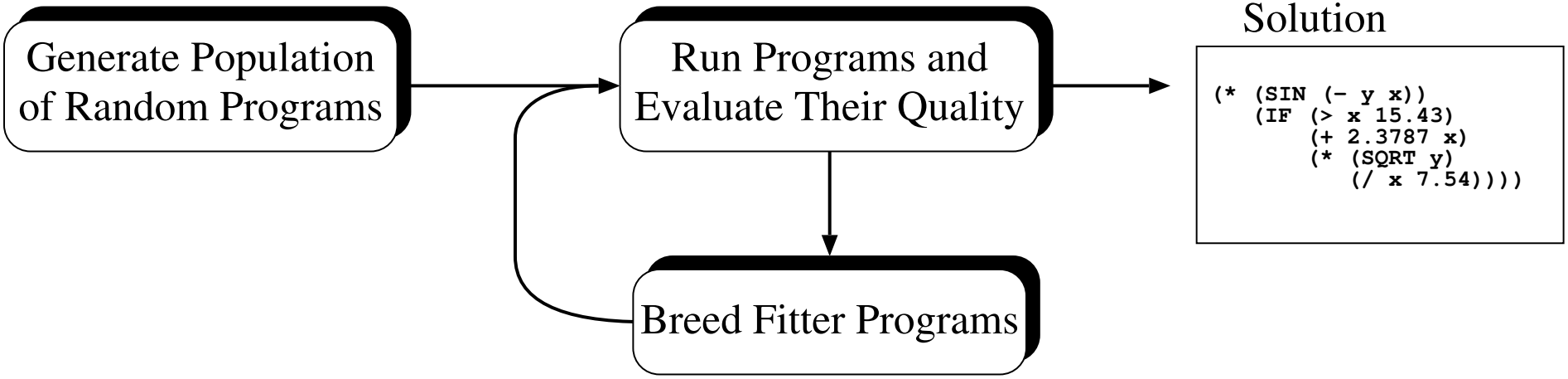
In a Nutshell

GP's randomness can escape traps which deterministic methods may be captured by

Like nature, GP is very successful at evolving novel and unexpected ways of solving problems

numerous examples will be shown later

In a Nutshell



Basic control for genetic programming: survival of the fittest is used to find solutions

Canonical genetic programming

1: Randomly create an *initial population* of programs from the available primitives (more on this in Section 2.2).
2: **repeat**
3: *Execute* each program and ascertain its fitness.
4: *Select* one or two program(s) from the population with a probability based on fitness to participate in genetic operations (Section 2.3).
5: Create new individual program(s) by applying *genetic operations* with specified probabilities (Section 2.4).
6: **until** an acceptable solution is found or some other stopping condition is met (e.g., a maximum number of generations is reached).
7: **return** the best-so-far individual.

In a Nutshell

GP finds out how well a program works by running it and then comparing its behavior to some ideal (line 3)

> E.g., how well a program predicts a time series or controls an industrial process

This comparison is quantified to give a numeric value called *fitness*

In a Nutshell

Programs that do well (i.e., have a good fitness) are chosen to breed (line 4) and produce new programs for the next generation (line 5)

Primary 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

Program

GP reasons about programs

GP takes programs as input, and produces better programs

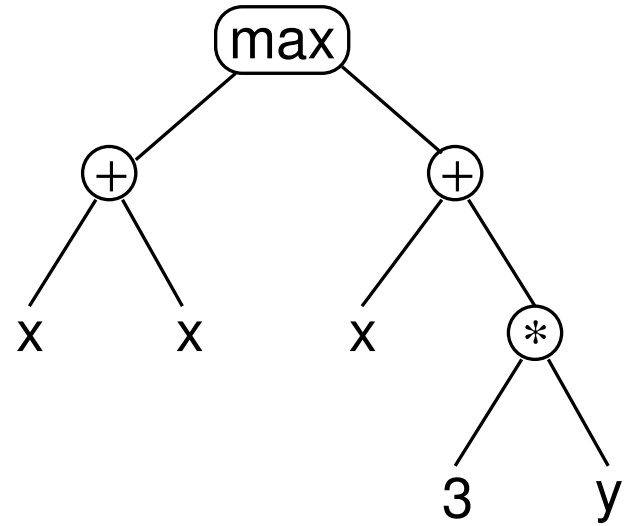
A program is meant to solve a particular problem

Program

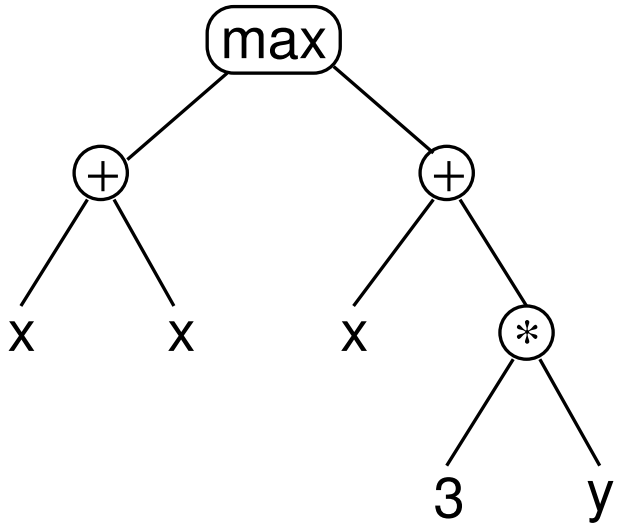
In GP, programs are usually expressed as Abstract Syntax Tree (AST)

For example, consider the following short program

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



Abstract Syntax Tree



Each element of the tree is a node

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

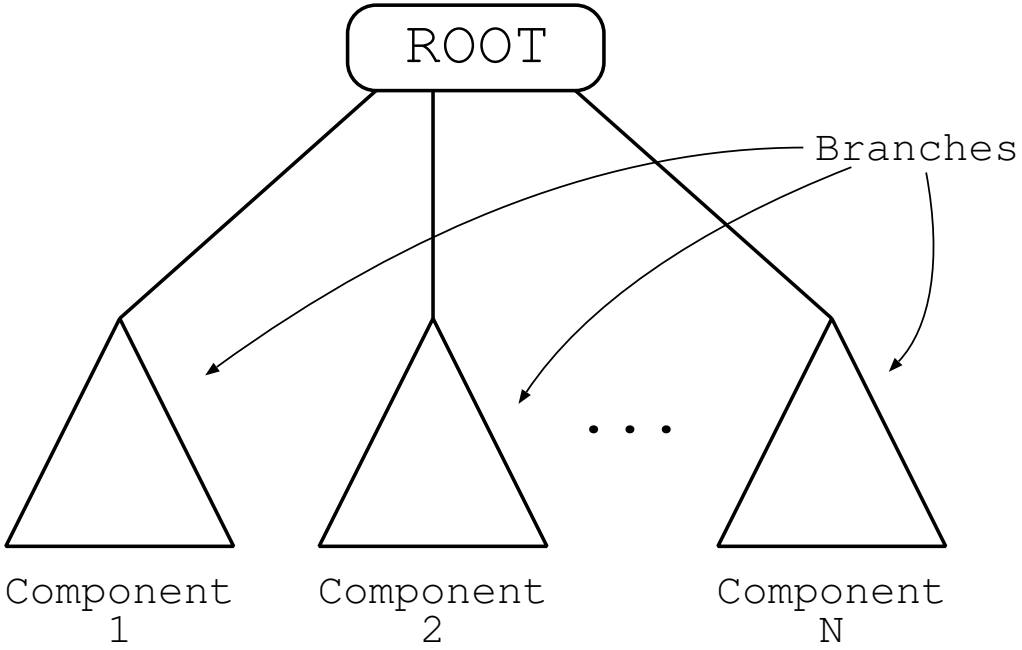
Internal nodes are called functions

functions + terminals = primitive set of a GP system

Program

In more advanced forms of GP, programs can be composed of multiple components (e.g., subroutines)

In this case, a program is a set of trees (one for each component) grouped together under a root node



Program

The notion of program is more general to what a Java programmer will consider

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

Instead, very specific languages are used

These languages are often very simple

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

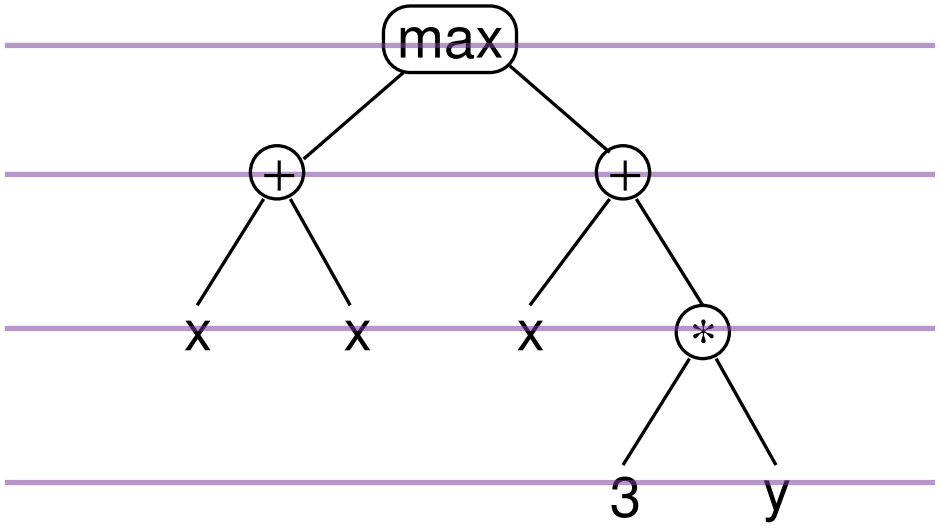
Initializing the population

The initial population is randomly generated

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



Depth

1

2

3

Selection

Genetic operators in GP are applied to individuals that are probabilistically selected based on their fitness

Better individual are more likely to have child programs than weaker individuals

The most commonly employed method for selecting individuals in GP is tournament selection, followed by fitness proportionate selection

Tournament selection

A number of individuals are chosen at random from the population

They are compared with each other and the best of them is chosen to be parent

When doing crossover, two parents are needed

So, two selection tournaments are made

When doing a mutation, only one parent is needed

Tournament selection

Note that tournament selection only looks at which program is better than another

- It does not need to know how much better
- this automatically rescales fitness

The selection pressure on the population remains constant

- A system with a strong selection pressure very highly favors the more fit individuals
- As a consequences, diversity of the population is reduced, causing serious problems

Tournament selection

As a consequence, a single extraordinarily good program cannot immediately populate the next generation with its children.

Tournament selection amplifies small differences in fitness to prefer the better program even if it only marginally superior to the other individuals in a tournament

Implementation

You need to provide the following functionalities on trees:

- Copy
- Printing
- Evaluation

Implementation

Create a small library to create trees

Two examples have to be implemented

Example 1: functions = {+ - * /} terminals = numbers

E.g., which equations involving a set of numbers solves a particular relation?

Example 2: functions = {+ - * /} terminals = x, y, z

E.g., which equations solves a particular problem? E.g., which equations accepts a set of roots