Genetic Algorithms and Programming

From majority classification to evolutionary program writing

Manuel Pita | May 2020

UNIVERSIDADE
LUSÓFONA

Today

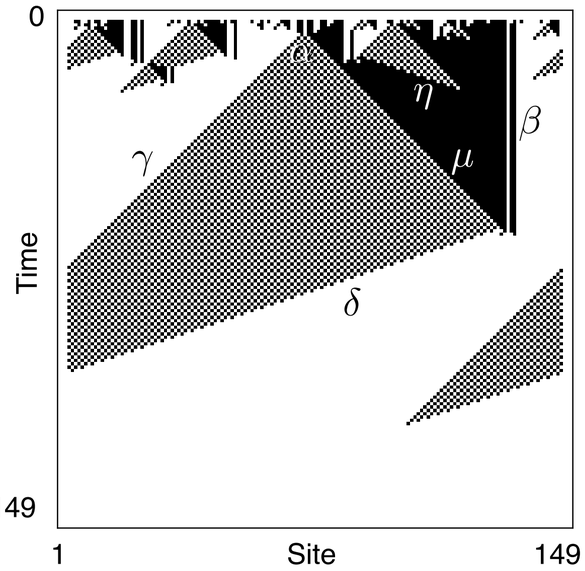
- General recap on Cellular Automata, Majority Classification
- Lets get a deeper view of genetic algorithms (GAs)
- Not a side story: Evolutionary Computation with Keith Downing at TEDx
- Step by Step guide to GAs and the majority classification task
- Introduction to Genetic Programming
- If time allows: Q&A

The Majority Classification Task

From CA rules, to collective information processing

- We have an **odd** number of automata
- Whatever random selection of of states there is always one majority state
- Can we figure out a CA rule that makes all automata end in the majority colour?
- What makes this a *complex problem*?
- What do we know about the majority classification task?
  - The rule needs to consider three (r=3) neighbours on either side
  - There is no perfect solution
- What is the number of possible rules we have in our search space?
- Don’t forget: the CA rule is the program, and the automata network is the input.
- The “program” runs for a number of time steps and the emergent pattern is the output
- **Do not confuse** CA rule (program) with automata configuration network (input)

GKL rule



The Basic Genetic Algorithm (again)

1. Generate a random population of solutions of size P
2. Compute fitness of each individual and sort population from highest to lowest fitness
3. Produce next generation
    A. Select the top E individuals and transfer them without change to the next generation
    B. Create remaining P-E individuals as children of ELITE
    C. Consider mutation for diversity
4. Repeat 2-3 until termination

Evolution and Computation

Not a side story: Ted talk by Prof. Keith Downing

- Keith Downing: Professor of Computer Science in Norway
- This is a bit of a longer video, 14 min
- Great summary of things we have been discussing over the last weeks
- As you watch the video, **identify the three ideas** that are most interesting to you
- If Zoom allows, poll

https://www.youtube.com/watch?v=D3zUmfDd79s

GAs and Majority Classification

Step 1: Generate a random population, of what?

- For the majority classification task we need a population of Cellular automata rules
- We know these CA rules need to have s=2 states and radius r=3
- Since n = 2r+1 + 7 our rule needs to include 2^7 entries
- That is 128 condition-action pairs
- Each action can be 0 or 1, so there are 2^128 possible different CA rules (or programs) we can try
- We want to search this space for CA rules that can perform the majority classification task well
- We start by producing P random different CA rules: our seed population (like Adams and Eves)
- Lets say P = 10

GAs and Majority Classification

Step 2: Compute Individual Fitness

- What is fitness?
- How good a rule does the majority classification.
- But how can we **quantify** that?
  - First decide how many automata are in your training networks, e.g. `N=91` (odd number)
  - Produce a random network initial configuration `init` with zeroes and ones
    - If `init` has more zeroes than ones, `majority = 0, else majority = 1`
    - Now run a individual CA rule from the population for a number of steps `t ~ 2N`
    - This will produce the space-time dynamics matrix lets call it `Phi`
    - Now take the last configuration of `Phi` and do
    - `performance = Phi.count(majority)/N`

GAs and Majority Classification

Step 2: Compute Individual Fitness. From one example to many

- Is it enough to compute the performance of one CA rule for **just one training example**?
- The answer is No: we need many examples to see if a CA rule we are considering performs well on average.
- Some CA rules may be very good at classifying when majority is 1, or when it is zero, etc.
- We want the best general rule we can find. What do we do?
- When you start your GA, create a set of training examples with many different initial configurations of size N
- Make sure examples are balanced, majority 0 and majority 1, you can also vary the size N
- Then compute the **performance** of your rule on all these examples and take the average to be the **fitness**

GAs and Majority Classification

Step 2: Compute Population Fitness and sort

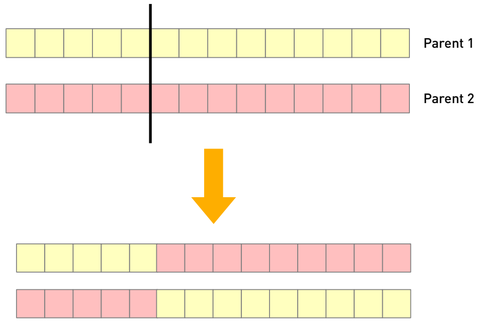
- We now have a procedure that computes the fitness of a CA given a number of examples.
- Do this for all the CA rules in your current population
- Keep the CA rules with their corresponding fitness in e.g. a list, dictionary or tuple
- Sort the population from the highest fitness to the lowest
- We have now completed the first big step of the algorithm!

GAs and Majority Classification

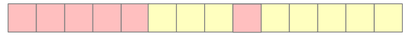
Step 3: Produce next generation

- We now have a population of CA rules ordered according to their fitness
- You will decide how many of the top CAs are transferred without change to the next generation
- These rules are considered to be the `Elite` the size of which is `E` (a parameter you set)
- What about the other P-E CA rules that we need to have a full population again?
  - Produce them by sexual reproduction of elite members (Crossover)
- Add some exogenous diversity by including some mutation for the children. Why?

Crossover



Mutation



GAs and Majority Classification

Step 4: Let evolution do its thing

- We repeat the production of new generations
- Typically this is done for many generations
- Or as we discussed last week, until populations reach some desired fitness
- The last generation of one run can be the initial one of another run you can do at a later point

GAs and Majority Classification

How to do this in Python? This is your project I

- First we need a function to **run a CA rule** on some automata network configurations for t steps
- **Study Numpy methods** that can allow you to run a CA in less time than with a vanilla soft algorithm
- **Do not forget that the automata near the edges** may have neighbours on the other side of the configuration list!
- Then you will need a **function to generate a random population of CA rules** with a given size, e.g. 128
- Also, a function to **generate a random set of X training examples of size N (automata configurations)**
- You need a **function to compute the majority classification performance** of a CA rule given a time-space matrix
- Another function to **compute fitness of a CA as average of performance** on set of training examples
- **Function to do crossover** given two parents and number of crossover points
- **Function to do mutation** given a child and mutation rate
- **Main function that orchestrates** of the steps needed to run a Generic algorithm
- For the programming geeks: **see how to make this run in the most efficient** way possible

Genetic Programming

How to evolve computer programs?

> How can computers learn to solve problems without being explicitly programmed? In other words, how can computers be made to what is needed to be done, without being told exactly how to do it? — attributed to *Arthur Samuel* in the 1950s

We have something similar: Inductive programming, learning from input-output examples. But this is very domain specific, not general. Used for scripting in excel, apple does it too. They are more about writing code not about designing solutions creatively. They are based on (combinatorial) search. We want a lot more than just macros.

GP: **Deliberate evolution** of computer programs for solving hard problems, for producing software, for understanding life

How? We supply tests for performance, programming blocks and operators

Same principles as Genetic Algorithms but with big difference: **Representation**

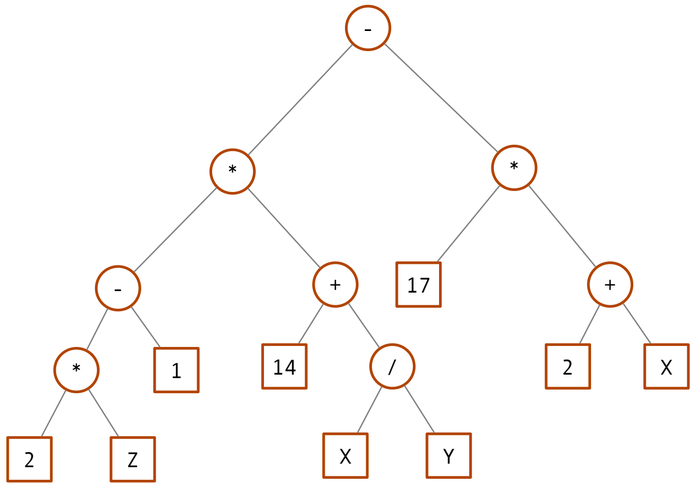
Genetic Programming

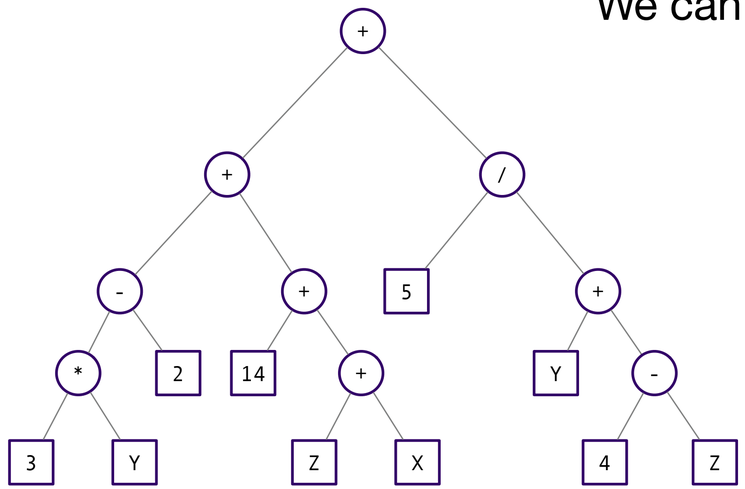
Representation (again)

```
Operators = [ +, -, *, / ]
Numbers = [0-9]
Variables = [X,Y,Z]

(17 * (2 + X) - (2Z -1) * (14 + (X / Y))

(- (* 17 (+ 2 X))
   (* (- (* 2 Z) 1)
      (+ 14 (/ X Y))))
```

We do not have fixed arrays of genes like in GAs. In Genetic Programming we work with operator trees that can have any structure

```
(+ (/ 5 (+ Y (- 4 Z))
   (+ (- (* 3 Y) 2)
      (+ 14 (+ Z X))))
```



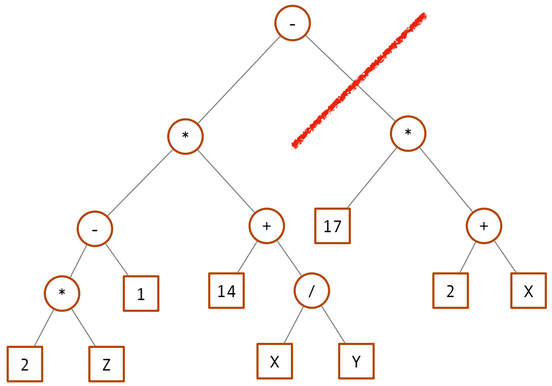


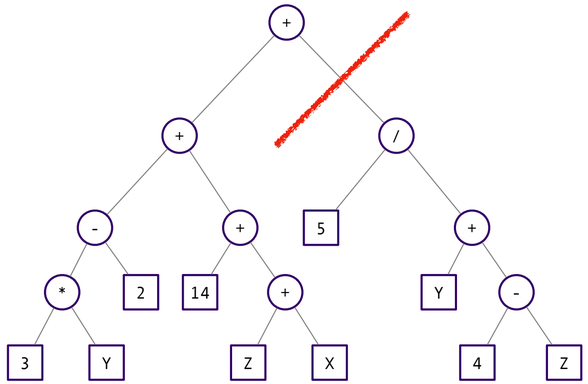
We can also represent programs like this

```
(if (< X 4)
   (while (> Z 0)
      (= Z (- Z 1)))
```

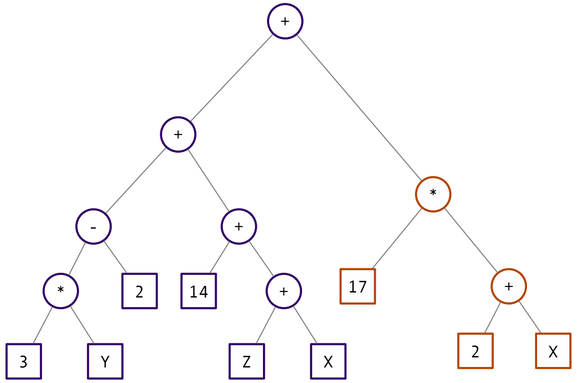
Genetic Programming

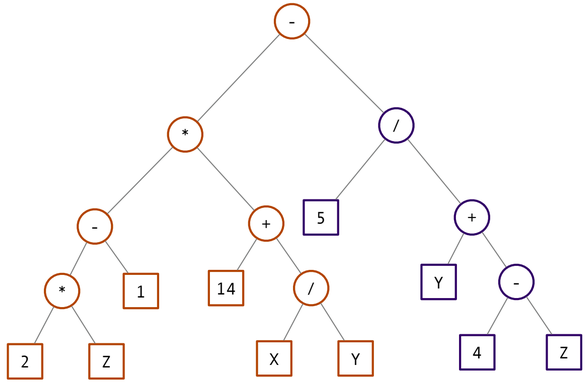
Representation (again) and crossover





Parents

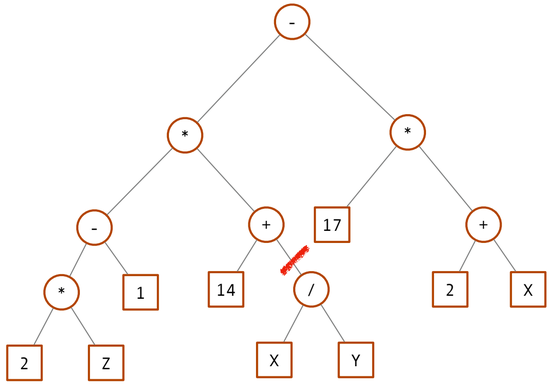


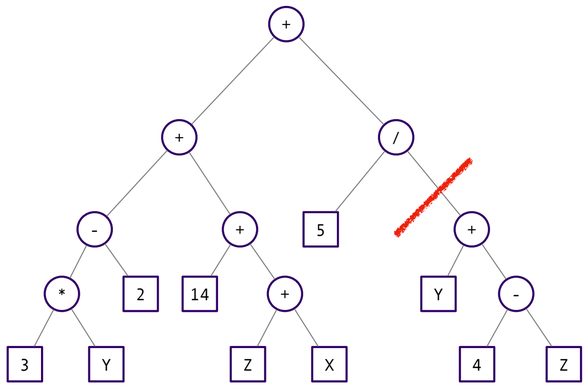


Children

Genetic Programming

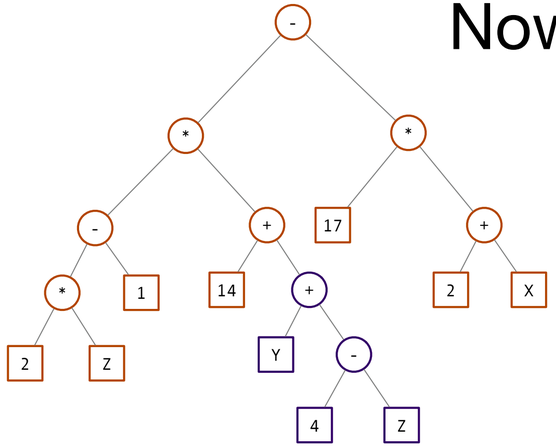
Representation (again) and crossover

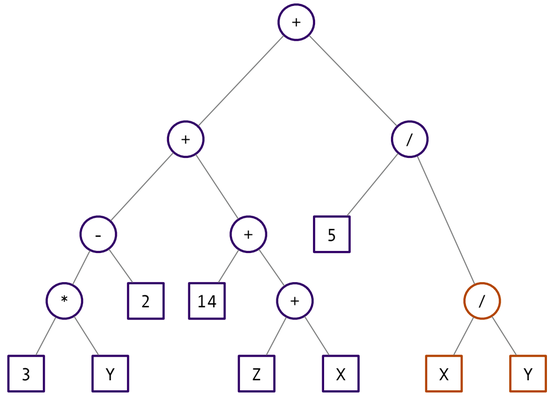




Parents

Now compute the children





That’s it for today