1 Introduction to Genetic Programming (GP)

Automatic generation of computer programs without explicitly programmed by humans.

Similarly with other evolutionary computing (genetic algorithms, evolutionary programming), genetic programming is:

- Inspired by (Darwinian) natural selection and genetics.

2 A Typical C Program

```c
void foo(int time, int a) {
    int temp;

    if (time > 10)
        temp = 5;
    else
        temp = 6;

    temp = temp + a;
    return temp;
}
```

3 A Typical Lisp Program

Lisp programs are S-Expressions (symbolic expressions) consisting of an atom or a list.

For example:

```
(+ a (IF (> time 10) 5 6))
```
Notice that prefix notation is used for Lisp programs.

The parse tree of a computer program (not necessarily in Lisp) is manipulated genetically using genetic programming. Programs in Lisp are in the form of a parse tree.

![Parse Tree of a Lisp Program](image)

Figure 1: The representation of a computer program as a parse tree.

4 Operation of Genetic Programming

1. Generate an initial population (e.g. random) of computer programs based on the chosen terminal set and function set.

2. Iteratively perform the following:
   
   (a) Execute every program in the population and assign it a fitness value, based on the fitness function.
   
   (b) Create a new population of computer programs (solutions) by applying probabilistically one of the genetic operators:
       - Reproduction
       - Crossover
       - Mutation

3. Designate the best individual so far, as the (approximate) solution to the problem.

5 The Terminal Set and Function Set of GP

The terminal and functional sets describe the components of a single computer program, synthesised by genetic programming.

For example, the program (solution)
(+ a (IF (> time 10) 5 6))

could use the terminal set:

Terminals = \{10, 5, 6, time\}

and the function set:

Functions = \{+, IF, >\}

Note that time is a variable containing a value.

6 Characteristics of the Terminal and Function Sets

The terminal and function sets should satisfy the closure property.

- Any value returned from a function in the function set, should be a valid argument to any other function in the function set.
- Any value from the terminal set, should be a valid argument to any function in the function set.

Note, that some common (arithmetic) operators might have to be redefined. For example, the protected division function takes two arguments, and it returns 1 if the denominator is 0, otherwise it returns the normal quotient.

7 Pseudocode of Genetic Programming

8 Genetic Operators used by GP

The most common genetic operators that genetic programming uses are:

- **Reproduction**: The process of selection an individual and copying it to the next generation. The individual is chosen based on its fitness, i.e. the higher its fitness, the higher the probability to be chosen.
- **Crossover**
- **Mutation**: In most cases mutation is not needed by genetic programming runs.

8.1 Crossover

The operation of crossover is illustrated in Figure 3. Two random points are chosen in the two parents and the corresponding subtrees are swapped, to produce the two offspring.

8.2 Mutation

Choose randomly a point in a tree and remove the subtree under that point. Create a random tree and insert it at that point.
Gen := 0

Create Initial Random Population

Termination criteria satisfied?

Yes
End

No
Evaluate fitness of each individual

i := 0

i := Gen + 1

i = M?

Yes

i := i + 1

No

Select Genetic Operation

Select one individual

Perform Reproduction
Copy into new population

Select two individuals

Perform Crossover
i := i + 1
Insert two offsprings into new population

Select one individual

Perform Mutation
Insert mutant into new population

Select one individual

Perform Reproduction
Copy into new population

Select two individuals

Perform Crossover
i := i + 1
Insert two offsprings into new population

Select one individual

Perform Mutation
Insert mutant into new population

Figure 2: Pseudocode for genetic programming.
Figure 3: The crossover operator used in genetic programming.
Figure 4: The crossover operator used in genetic programming.
9 The Ramped half and half generative method

The method which has been found to work better for the generation of the random initial population is the *ramped half-and-half* method.

According to this:

- Half of the individuals of the initial population are generated having the maximum allowed initial depth (typically 6).
- The rest of them will have a depth from 2 to the maximum allowed depth (e.g. 6).
  - 20% of them will have depth 2.
  - 20% of them will have depth 3.
  - 20% of them will have depth 4.
  - 20% of them will have depth 5.
  - 20% of them will have depth 6.

10 Applications of Genetic Programming

Some typical applications of GP are:

- Regression
- Optimal Control
- Forecasting
- Discovering Game Strategies
- Electrical Circuit Design
- ...

11 Problem 1 - Artificial Ant

Navigate an artificial ant in a maze so as to find all food. This is illustrated in Figure 5.

The terminal set for this problem is:

\[ T = \{\text{MOVE, RIGHT, LEFT}\} \]

The function set is:

\[ F = \{\text{IF-FOOD-AHEAD, PROGN2, PROGN3}\} \]

where function \( \text{PROGN2} \) executes its arguments in sequence, e.g. \( \text{PROGN2}(\text{LEFT})(\text{RIGHT}) \) instructs the ant to first turn left and then right.

Function \( \text{PROGN3} \) executes its three arguments in sequence.
Figure 5: The artificial ant problem for the Santa Fe trail.
11.1 Parameters for the Artificial Ant Problem

The following parameters were chosen for running GP in the artificial ant problem:

1. **Success Predicate**: A symbolic expression passing through all the 89 pieces of food.
2. **Fitness cases**: One fitness case (a fixed start position for the Santa Fe trail).
3. **Raw fitness**: Number of pieces of food before the ant times out with 400 iterations.
4. **Standardised fitness**: 89 - raw fitness.
5. **Population size**: 500.
6. **Maximum number of generations**: 51.
7. **Probability of reproduction**: 0.1.
8. **Probability of crossover**: 0.9.
   
   90% of the time crossover points are chosen within the internal nodes of the tree, and the rest of the times the crossover is performed at a point in the leaves of the tree.
9. **Maximum depth of initial population tree**: 6.
10. **Maximum depth of trees created by the crossover**: 17.
11. **Initial population generated using**: ramped half and half method.

In generation 21, the following S-expression was found, able to score 89 out of 89:

```
(IF-FOOD-AHEAD (MOVE)
  (PROGN3 (LEFT)
    (PROGN2 (IF-FOOD-AHEAD (MOVE)
      (RIGHT))
      (PROGN2 (RIGHT)
        (PROGN2 (LEFT)
          (RIGHT))))
    (PROGN2 (IF-FOOD-AHEAD (MOVE)
      (LEFT))
      (MOVE))))
```

12 Problem 2 - Cart Centering

The cart centering problem involves a cart that can move to the left or right on a frictionless one-dimensional track.

The problem is to center the cart, in minimum time, by applying a force of fixed magnitude, so as to accelerate the cart towards the left or the right.

- There are two state variables for the system, the position $x(t)$ and the current velocity $v(t)$.
There is one control variable, the direction of the force $F$ to the centre of mass of the cart, so as to accelerate the cart in either the positive or negative direction.

- Target: The state where the cart rests (velocity 0.0) at position 0.0.

At time $t + 1$ the velocity of the cart is given by:

$$v(t + 1) = v(t) + \delta t \cdot a(t)$$  \hspace{1cm} (1)

where $\delta t$ the time step and $a(t)$ the acceleration given by:

$$a(t) = \frac{F(t)}{m}$$  \hspace{1cm} (2)

The position of the cart $x(t + 1)$ is:

$$x(t + 1) = x(t) + \delta t \cdot v(t)$$  \hspace{1cm} (3)

The exact time-optimal solution has been found to be the one that apply force $F(t)$ to accelerate the cart to the positive direction if:

$$-x(t) > \frac{v(t)^2 \text{sign}[v(t)]}{2|F|/m}$$  \hspace{1cm} (4)

or otherwise apply $F$ so as to accelerate the cart towards the negative direction.

For the purposes of the GP simulations, we assume that $m = 2.0$ and $|F| = 1$ therefore the optimal (unknown for the GP) control strategy is to push the cart towards the positive direction, if:

$$-x(t) > v(t)^2 \text{sign}[v(t)]$$  \hspace{1cm} (5)

### 12.1 Terminal and Function sets for the Cart Centering Problem

The chosen terminal set is:

$$T = \{X, V, -1\}$$

The chosen function set is:

$$F = \{+, -, *, /, \text{GT}, \text{ABS}\}$$

where $/$ is the protective division operation. $\text{GT}$ is the greater-than function and returns 1 for true or -1 for false.
12.2 Parameters for the Cart Centering Problem

The following parameters were chosen for running GP in the cart centering problem:

1. **Objective**: Find an optimal bang-bang control strategy to center the cart at position 0 with velocity 0.

2. **Fitness cases**: 20 initial condition points \((x, v)\) for position and velocity chosen randomly from the square in \(x, v\) space whose opposite corners are \((-0.75, 0.75)\) and \((0.75, -0.75)\).

3. **Raw fitness**: Sum of the time, over the 20 fitness cases, taken to center the cart. When a fitness case times out, the contribution is 10.0 sec.

4. **Standardised fitness**: Same as raw fitness.

5. **Population size**: 500.

6. **Maximum number of generations**: 51.

7. **Probability of reproduction**: 0.1.

8. **Probability of crossover**: 0.9.

   90% of the time crossover points are chosen within the internal nodes of the tree, and the rest of the times the crossover is performed at a point in the leaves of the tree.

9. **Maximum number of time steps**: 500.

10. **Maximum depth of initial population tree**: 6.

11. **Maximum depth of trees created by the crossover**: 17.

12. **Initial population generated using**: ramped half and half method.

13. **Wrapper**: Converts any positive value returned by an S-expression to +1 and converts all other values (negative or 0) to -1 (negative forces).

In generation 33 the best individual found is the optimal solution to the problem:

\[
(- (- (* (+ (* (ABS V) -1) (* (ABS V) -1)) V) X) X)
\]

The above expression is equivalent to the optimal strategy:

\[
(GT (* -1 X) (* V (ABS V)))
\]
13 Automatic Generation of Constants for the Terminal Set

In many problems, it is not possible to predict what constants would be useful in the terminal set.

In such cases (which is the common case), the terminal set can be augmented to include a random constant $R$ which corresponds to a random number within an appropriate range and appropriate resolution (e.g. we might only be interested in integer random numbers, in a specific range).

- The initial population will contain a variety of random constants.
- Once a constant is inserted in a S-expression is remains fixed and propagates through individuals with the genetic operations.