COMP 572

Project #2a – Genetic Programming

Sanjeev Shrestha

March 03, 2014

This is the first subproject of the GP project. The goal of this subproject is to create a population of GP tree structures for a symbolic regression problem.

If you want to use it, or refer to it, I have written a node and an individual class that uses pointers to build and evaluate random expression trees. Trees are built of nodes, which point to each other.

For this subproject you only need the following functionality:

- Generate full random expression individuals.
- The expression trees should have, at least, the non-terminals: +, -, *, /.
- The expression trees should have, at least, the terminals: X (the input variable) and constants.
- The ability to copy individuals.
- The ability to evaluate individuals.
- The ability to erase individuals.
- The ability to calculate the size (number of terminals and non-terminals) of individuals.
- The ability to create a population of individuals and to find the best and average fitness of the population, and the average size of the individuals in the population.

Individuals should represent expression trees, but may be coded as a different type of data structure (e.g. a tree stored in an array). For now you may choose your own fitness function, i.e. your own set of x,y points that the GP should evolved an expression to fit.

1. Algorithm Description

Genetic Programming (GP) is a branch of artificial intelligence that is meant for creating a working computer program to solve a user defined problem or task [1]. Genetic Programming works on a set of computer instructions, the fitness function here is simply defined by how well those instructions can achieve a certain computational task [3] [4] [5]. Genetic Programming (GP) is a specialized view of genetic algorithms (GA) where each individual is represented by a computer program.

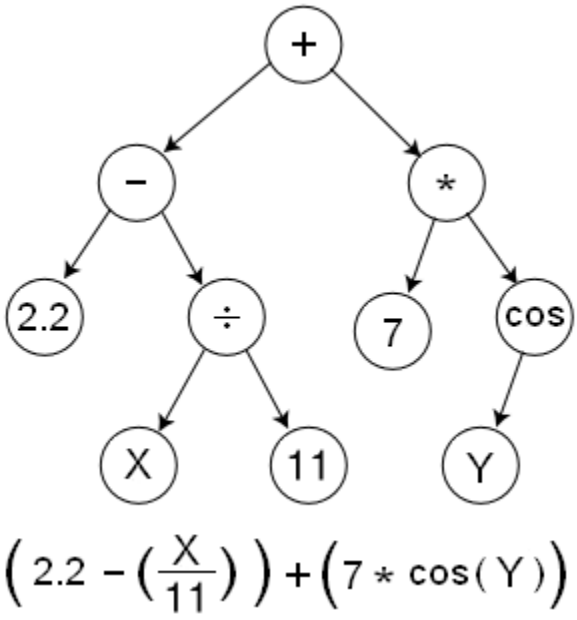

Fig: A Function represented as a tree structure[2]

For solving this problem, I have setup the Individual, the Node and the Population Class. I have also setup different classes for each of the Commands or Operators that will be used in the expression tree. I have also created a Driver Class which will be used to create and test the population against a particular set of input and output data. Other specialized classes are designed for gathering Population related data, Constants and excel file generation. Further details of the work are outlined below:-

2. Node Description

The Node class implements the Serializable class. As everything in Java is passed by reference, the copies are truly not independent of each other until and unless we make a deep copy of the object. Hence to make a separate copy of the Individuals which are truly separate from each other, I make the Node class and Individual class serializable. I then use the Apache Commons Lang SerializationUtils class to make a deep copy of the Individual.

The attributes of this class are:-

- Parent: to keep a track of the root node and used in recursion

- Branches [4]: Four Branches are initialized that store a different Node.
- Type: Type of Node, either it is Operator (ADD, SUBTRACT, MULTIPLY, DIVIDE) or Operand (CONSTANT VALUE, INPUT VARIABLE X).
- ConstValue: Value stored by terminal node for constant value.
- Terms: Number of Terminal Nodes
- NonTerms: Number of Non Terminals
- Expression: The random expression represented by the tree. Expression generated by in order traversal.

The corresponding methods used in this class are:-

- Public void full(int depth, int max, Node parent): Recursively generated a tree until a certain height has been reached. “Depth” represents the current depth of the tree, “max” represents the maximum height and “parent” represents the current parent node during recursion.
- Public float evaluate(float X): Recursively evaluates the value generated by the tree.
- Public int calcSize(): Calculates the size of the tree and stores them in terms and nonTerms.
- Public String iterativeInorder(Node root): Generates the expression in the tree by traversing the tree in in order fashion.
- Public String getType(Node): Returns the corresponding Character as seen in the Node. For example: “return +” for Node Type ADD.

3. Individual Description

The Individual class consists of a Node value (this is the root of the tree), a float value that represents the calculated fitness and an integer value for the size of the tree (sum of number of terminals and non-terminals). Following is an overview of the different methods used:-

Public void evaluate(): Evaluates the fitness of the tree.
Public void evaluatePrint(): Prints the evaluation of the tree
Public void calcSize(): Calculates the size of the tree (sum of terminals and non-terminals)
Public void generate(int): Generate the Individual and assign the root node.
Public String toString(): Evaluates the expression of the Individual in In order fashion.

4. Population Description

The population class consists of a Set of Individuals, a subset population selected by selection mechanism to apply mutation and crossover (for later functionality), a set of PopulationStatistics objects, a value to store average size of the population, a string to get the function name as to which various calculations are branched out, the population size, offspring size, tree height and the Boolean flag to find if solution has been found and an Individual solution to the given problem. The methods in this class are:-

- generatePopulation(): to generate the Population until the population size has been reached.
- selectedIndividuals(): returns the individuals who have been selected.
- checkForSolution(): check if possible solution has been found as per the threshold.
- getTournamentWinner(): to get the Individual who has won the tournament.
- getRandomIndividual(population): get a random individual from the population.
- appendToPopulation(populationSubSet): add the offsprings to the original population
- removeExtraPopulation(): remove the individuals with low fitness as to maintain the population size.
- populationReset(): remove all individuals from the population.
- evolve(): evolve the population.

  The evolve method evolves the population as to one generation. The evolve method is invoked from the GPTestClass.

5. Results

- Test for Deleting Individuals:
  - Population Size : 25
  - Individual to Remove : 6.596483 * 5.2512083
  - Population Size : 24

- Test for Copying Individuals:
  - Original Status of Source and Clone
  - Source Individual: X / 12.087288
  - Destination Individual: X / 12.087288

  - Status after changes made to Destination
  - Source Individual: X / 12.087288
  - Destination Individual: X / 3.0

  - Status after changes made to Source
  - Source Individual: X / 9.0
  - Destination Individual: X / 3.0

- The results on basis of Tree Height versus Average Fitness and Best Fitness generated are given below:-

Tree Height	Average Fitness for Generation	Average Size for Generation	Time to evolve (in microseconds)	Best Individual's Fitness	Best Individual For Generation
1	29.42	3	68	0	X * X
2	76.96	7	55	15.86278534	-4.1024675 / X + 12.983822 + X
3	175.17	15	57	16.51752663	5.1509943 - 11.657274 + -4.3591022 / X * - 1.6641903 / X + 3.207841 - X
4	3436.7	31	56	15.97063255	X - X / X * 0.03895378 + X + X + X / X - X / 5.376958 / X * X - X - X / -1.291064 + X

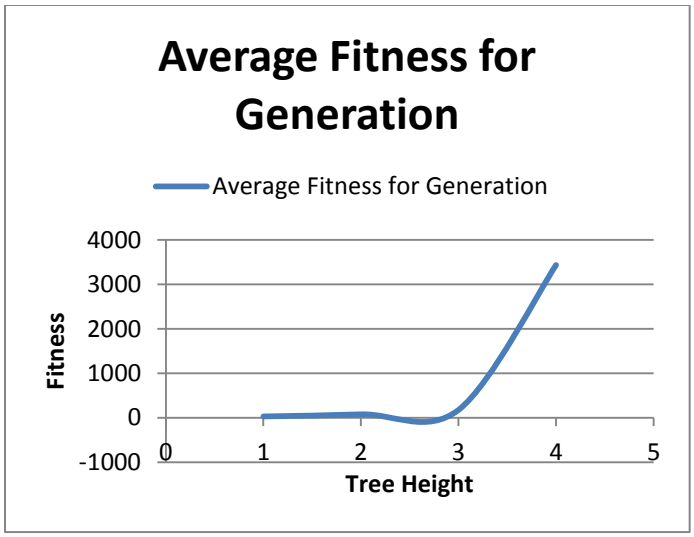

Fig: Tree Height vs. Average Fitness

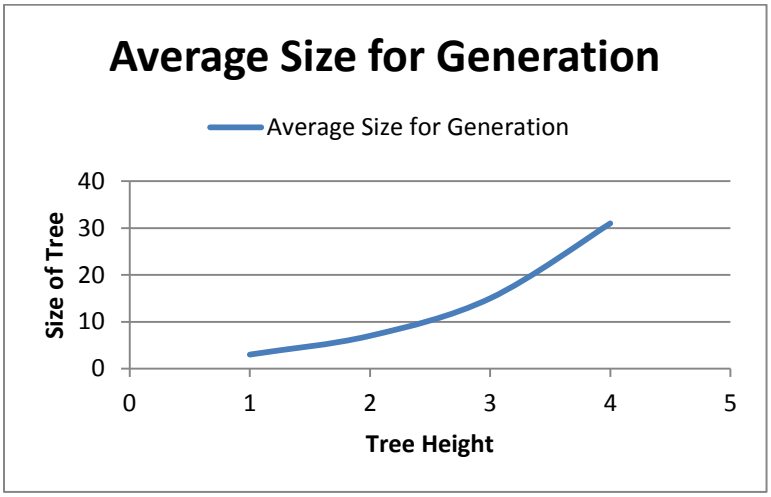

Fig: Tree Height vs. Size of Tree

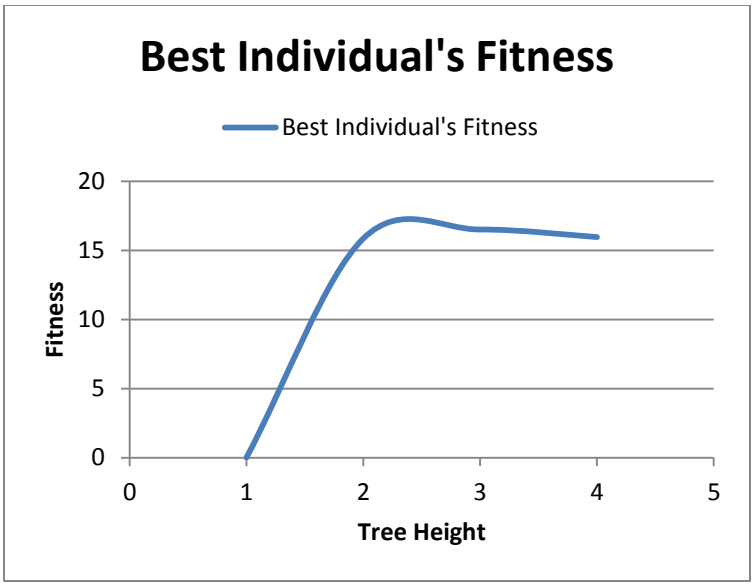

Fig: Tree Height vs. Best Individual's Fitness

6. Conclusion

In conclusion, Genetic Programming can used to generate random expression tress based on the operator selection and constant values. I found this approach to be very intuitive because it combined the aspects of Computer Science and Statistics very closely. The idea of Symbolic Regression to find a formula that best fits the data points is best outlined by this project.

Works Cited

1. Eiben, Agoston E., and J. E. Smith. "Genetic Programming." *Introduction to evolutionary computing*. New York: Springer, 2003. 101. Print.
2. "Genetic programming." *Wikipedia*. Wikimedia Foundation, 16 Feb. 2014. Web. 3 Mar. 2014. <http://en.wikipedia.org/wiki/Genetic_programming>.
3. Koza, John R.. "5." *Genetic programming*. 3. print. ed. Cambridge, Mass. [u.a.: MIT Press [u.a.], 1993. 73-75. Print.
4. Soule, Terence. "Genetic Program." *GP2A*. N.p., n.d. Web. 1 Mar. 2014. <http://www2.cs.uidaho.edu/~cs472_572/s14/GPProjectA.html>.
5. "Welcome to Genetic Programming." *genetic-programming.com-Home-Page*. N.p., n.d. Web. 4 Mar. 2014. <http://www.genetic-programming.com/#_What_is_Genetic>.