An Approach to Geometric Modelling Using Genetic Programming

Snigdhajyoti Ghosh, Damodar Goswami and Chira Ranjan Datta
Department of Electronics & Communication
Netaji Subhash Engineering College
Garia, Kolkata
snigdhajyoti.ghosh@gmail.com, goswami.damodar@gmail.com, crdatta@gmail.com

Abstract. In this work we 'derived' the famous Pythagorean theorem from the measurements of the sides of right-angled triangles with machine learning. In classical Euclidean geometry this result is proved with rigorous geometrical argument but we have followed a data driven approach and got the same result without entering a single step in the domain of geometry. We used symbolic regression with Genetic Programming to reach the model. As far as our knowledge goes, this result is a novel one and may open up a new avenue of applying machine learning tool in Geometry.

We have used python programming language 3.7 and libraries such as DEAP (v1.2), pygraphviz. The whole project can be found on https://github.com/snigdhasjg/Pythagorean-Triplate.git

Keyword: Genetic Programming, DEAP, Genetic Algorithm, Pythagorean Triple

1 Introduction

Genetic Programming is a Machine Learning tool that is driven by the evolutionary principle. Based on that principle it tries to find patterns in the data automatically without human intervention. Since evolution has become successful in sustaining and developing life on the earth, it is hoped that it can also solve many of our real-life problems. Genetic Programming is an extension of Genetic Algorithm which is tree based and can generate mathematical functions by evolving on its own. In other word Genetic programming is a special field of evolutionary computation that aims at building programs automatically to solve problems independently of their domain.

Symbolic regression is a type of regression analysis that searches the space of mathematical expressions to find the model that best fits a given dataset, both in terms of accuracy and simplicity.

There are many software to implement GP. First, we started experimenting with 'GPLAB', a MATLAB based software developed by Sara Silva. But eventually we moved to Distributed Evolutionary Algorithms in Python abbreviated as 'DEAP'. The main reason behind this is that Python is an open source language and is steadily upcoming. In Python, there are also more than one software. Among them we finally chose 'DEAP' because it is the most flexible and have a wide community support.

Our motive is to examine whether we can 'prove' the celebrated Pythagoras theorem without having the domain knowledges such as the properties of triangles, axioms of geometry, way of geometrical inferencing etc, just from the numerical measurements of the sides of triangles applying a machine learning algorithm. we have used Genetic Programming for this purpose.

1.1 Genetic Algorithm and related works

Genetic Algorithm (GA) is a bio-inspired optimization technique, first introduced by Holland [1]. GA has five main components, namely, chromosome encoding, crossover, mutation, evaluation and selection. Traditional structure of GA is shown in Fig. 2.1 and it is generally described as follows: "Genetic algorithm ... starts with an initial set of random solutions called population. Each individual in the population is called a chromosome representing a solution to the problem at hand... The chromosomes evolve through successive iterations, called generations. During each generation, the chromosomes are evaluated using some measures of fitness. To create the next generation, new chromosome, called offspring, are formed by either (a) merging two chromosomes from current generation using a crossover operator or (b) modifying a chromosome using a mutation operator. A new generation is formed by (a) selecting, according to the fitness values, some of the parents and offspring and (b) rejecting others so as to keep the population size constant. Fitter chromosomes have higher probabilities of being selected. After several generations, the algorithms converge to the best chromosome, which hopefully represents the optimum or suboptimal solution to the problem" [2].

In genetic algorithm there are these steps [3]

A. [Start] Generate random population of n chromosomes (suitable solutions for the problem)
B. [Fitness] Evaluate the fitness f(x) of each chromosome x in the population
C. [New population] Create a new population by repeating following steps until the new population is complete
   a. [Selection] Select two parent chromosomes from a population according to their fitness (the better fitness, the bigger chance to be selected)
   b. [Crossover] With a crossover probability cross over the parents to form a new offspring (children). If no crossover was performed, offspring is an exact copy of parents.
   c. [Mutation] With a mutation probability mutate new offspring at each locus (position in chromosome).
   d. [Accepting] Place new offspring in a new population

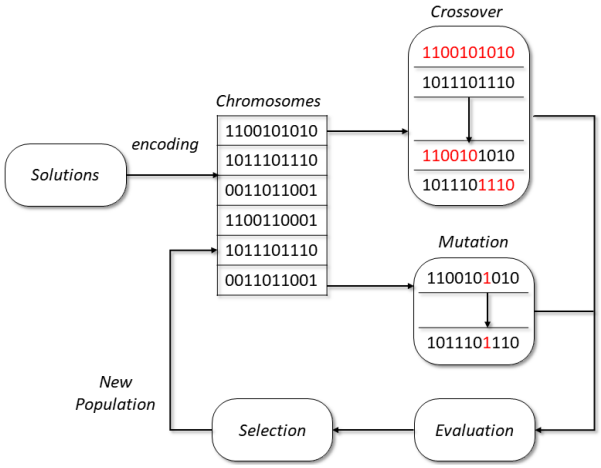

Fig. 2.1: Traditional structure of GA

D. [Replace] Use new generated population for a further run of algorithm
E. [Test] If the end condition is satisfied, stop, and return the best solution in current – population
F. [Loop] Go to step B

1.2 Genetic Programming and related works

Genetic Programming is a Machine Learning tool that is driven by the evolutionary principle. Based on that principle it tries to find patterns in the data automatically without human intervention.

Since evolution has become successful in sustaining and developing life on the earth, it is hoped that it can also solve many of our real-life problems. Genetic Programming is an extension of Genetic Algorithm which is tree based and can generate mathematical functions by evolving on its own. In other word Genetic programming is a special field of evolutionary computation that aims at building programs automatically to solve problems *independently of their domain*.

1.2.1 Application of GP

GP has been successfully used as an automatic programming tool, a machine learning tool and an automatic problem-solving engine. GP is especially useful in the domains where the exact form of the solution is not known in advance or an approximates solution is acceptable (possibly because finding the exact solution is very difficult). Some of the applications of GP are curve fitting, data modelling, Symbolic regression, feature selection, classification, etc. John R. Koza mentions 76 instances where Genetic Programming has been able to produce results that are competitive with human-produced results. These human-competitive results come from a wide variety of fields, including quantum computing circuits, analog electrical circuits, antennas, mechanical systems, controllers, game playing, finite algebras, photonic systems, image recognition, optical lens systems, mathematical algorithms, cellular automata rules, bioinformatics, sorting networks, robotics, assembly code generation, software repair, scheduling, communication protocols, symbolic regression, reverse engineering, and empirical model discovery. This paper observes that, despite considerable variation in the techniques employed by the various researchers and research groups that produced these human-competitive results, many of the results share several common features. Many of the results were achieved by using a developmental process and by using native representations regularly used by engineers in the fields involved. [4] Other than that GP is vastly used in Data mining in the medical field.

1.3 Symbolic Regression and related works

Symbolic regression is a type of regression analysis that searches the space of mathematical expressions to find the model that best fits a given dataset, both in terms of accuracy and simplicity. GP is the only tool for Symbolic regression.

Neural Networks (NN), Support Vector Machine (SVM), and GP have advantages over classical statistical methods in cases where no or negligible a prior information is known about the process and no assumptions on models can be made and where modelling problems are multidimensional with either too much or too little data. [5]

The benefits of symbolic regression via GP include the following. [6]

- There are no prior assumptions on model structure.
- The final predicting model or model ensemble is chosen from a rich set of nonlinear empirical models that is generated automatically.
- Sensitivity analysis of the inputs and variable selection is implicitly performed with no extra cost, which reduces the dimensionality of the problem.
- No assumptions are made on independence of input variables.
- Insight can be provided due to the symbolic representation of models (e.g., in a form of low-order variable transformations that can be used as “natural” meta-variables).

2 Methodology

We have a set of right-angled triangles and length of each sides of those triangles. So, in numerical terms, we have a set of triplets and we have to extract the relationship among the three sides represented by the triplets.

2.1 Configuration of DEAP Software

2.1.1 Input Preparation

We start with 200 triplets of (a, b, c) where (a, b, c) are the three sides of a right angled triangle with the hypotenuse c. The Pythagorean theorem tells us the relation between these three variables is $c\hat{}2 = a\hat{}2 + b\hat{}2$. We try to see whether the same relation can be found out with Genetic Programming only. For this purpose, we tried to establish the equation as $c\hat{}2 - (a^2 + b^2) = 0$, so technically we are searching for a function of (a, b, c) where the function returns 0. But the evaluation process does not know much about what function should it return rather than a function which has fitness score near a threshold (In this case it is 0). So, it was returning a function like $f\ (a, b, c) = a - a$ or $f(a, b, c) = b - b$ etc. This type of function isn't acceptable because with any value of a, b or c it always returns 0.

Then we moved on to $c = f(a, b)$. Here the inputs are 'a' and 'b' and for every unique set of (a, b) we are getting a different 'c'. The fitness criteria is to match $f(a, b)$ with respective 'c'.

2.1.2 Creating Primitive Set

Primitive set refers to a set of operator which will be used for construct the output function. So a set of basic math operator (i.e. Addition, Subtraction, Multiplication, Division, etc.) can be used as primitive set. The leaf or terminal node are consists of random numbers and input variables.

We have used Strongly typed GP where every primitive and terminal is assigned a specific type. The output type of a primitive must match the input type of another one for them to be connected. For example, if a primitive returns a Boolean, it is guaranteed that this value will not be multiplied with a float if the multiplication operator operates only on floats.

Choosing primitives set is one of the most crucial aspect of a GP. In this problem, we use a classical set of primitives, which are basic arithmetic functions (i.e. Addition, Subtraction, Multiplication, Division and Power). We have created our own Division for overcoming 'ZeroDivisionError' and Power for taking rational exponents. And added a 'Terminal' of multiple value i.e. called 'EphemeralConstant'.

2.1.3 Preparation of Toolbox and Statistics

Individual Creation: As any evolutionary program, symbolic regression needs (at least) two object types: an individual containing the genotype and a fitness. Genotype in simple word is each node of a tree where the tree itself is an individual. An Individual should have set of rule and set of parameter. Next we added 'Toolbox' for evolution that contains the evolutionary operators (i.e. selection, crossover and mutation). With 'Toolbox' we can register how a selection, crossover and mutation will happen and can decorate as needed. Fitness Function (also known as the Evaluation Function) evaluates how close a given solution is to the optimum solution of the desired problem. It determines how fit a solution is. Each problem has its own fitness function. For the fitness function we have used mean square error (MSE).

For statistical calculations we used module 'numpy' and evaluated standard minimum, standard maximum, mean, standard deviation.

2.1.4 Launching the evolution:

At this point, DEAP had all the information needed to begin the evolutionary process, but nothing has been initialized. We started the evolution by creating the population and then calling a complete algorithm.

2.2 Extension of DEAP Software

We needed to modify the main code of the software to modulate the stopping criteria. the default setting is based on the number of generations. The program will stop running when the given number of generations is exhausted. But we wanted to modify it. We wrote a small piece of code so that it stops when a particular fitness has been achieved.

With simple GP function it is difficult to achieve global optima. A basic multi-start procedure can help GP improve the probability of jumping out of the local optima and finding the global optimal solution [7]. This procedure starts the algorithm multiple times
and then pick the best solution among those found over all runs [8-10].

2.3 Experimentation

First, we took 50 data point, then the success rate was very low. It was taking too much time to collect the optima, most of the time it converges to local optima instead of global optima. The probability of jumping out from local optima was low because the input points was not enough.

So, then we move on to 500 points. In this case the computation time for each generation became very high. After many runs, we figured out for no of points around 200-300, the algorithm was not taking too much time for computation and able to jump out from local optima as it able to see more vector space.

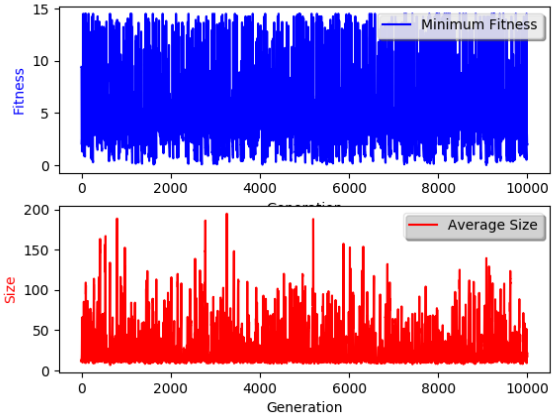

Fig 3.1

We took minimal approach to search for solution. We decorate the mate and mutate method to limit the size (size can be expressed as number of node in the tree representation of individuals) of generated individuals. This is done to avoid an important drawback of genetic programming: bloat [11-12]. Koza in his book on genetic programming suggest to use a max depth of 17 [13]. So, we started from tree depth 3 as this is the minimal tree need for this problem. Then we gradually move on to 4 and 5. Then we stick to 5 because grater tree depth lead to more complex tree. Also, this has the dual benefits of providing the simplest/smallest solutions and preventing GP bloat thus shortening run times. GP search should be limited to program lengths that are within the limit and that can achieve optimum fitness[14]. In fig 3.1 the maximum tree depth is 17 and it runs for 10000 generation and find a solution which has fitness value 0.0117 near our optimum fitness i.e. less than 0.01, but the tree became very complex. The average tree size over 10000 generation is 27.997. On the other

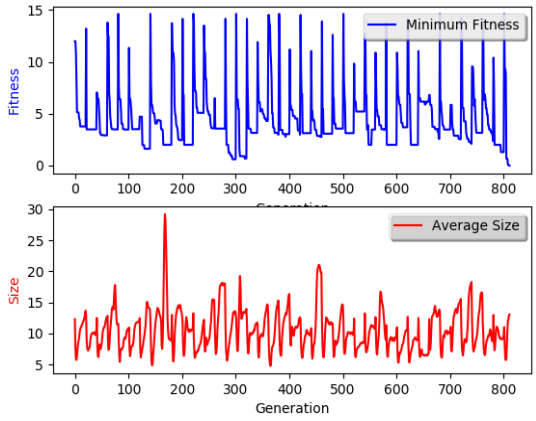

Fig 3.2

hand, in fig 3.2 the maximum tree depth is 5 and it runs 811 generation and able finds solution. The average tree size over 811 generation is 10.511. This height limit leads to less complex tree.

Okkam's razor principle states: "No more things should be presumed to exist than are absolutely necessary." Following this principle, we should limit and control the complexity of models we create and favour the simplest ones to take part in the evolution. Construction of an unknown function in a high-dimensional space from a finite number of samples bears the risk of over-fitting.

We fixed a resetting point after 20 generation. As we observed the maximum time the fitness stuck around 10th generation (In fig 4.1 the model converges after 10th gen). So, every after 20th generation the population get reset. With that we got success in finding solution.

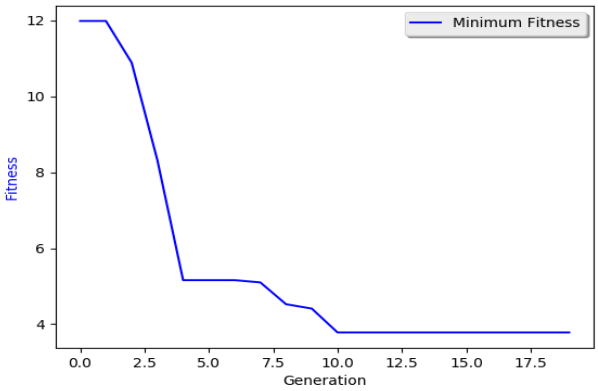

Fig 4.1: Generation vs Fitness

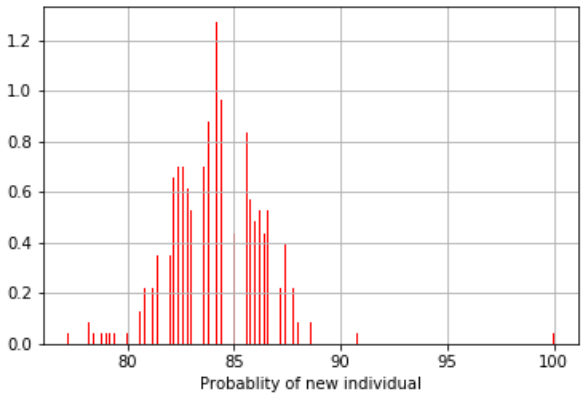

Fig 4.2: Effects of selection Crossover and Mutation

In fig 4.2 we clearly see the effects of Selection, Crossover and Mutation in the new Population. We have used tournament-based selection (tournament size 5) with 80% crossover probability where crossover happens via exchanging subtree with the point as root between each individual and uniform mutation happens at rate of 20%. In the above figure we can see that almost 85% of the population changes in the next generation.

3 Result

With this approach, we have achieved 8 successful runs from 10 runs. And got results like $c = c = \sqrt{A^2 + B^2}$, $c = \sqrt{A^2 + B^2 - B^{-7/5}}$, $c = \sqrt{A^2 + B^2 + \frac{B-B}{A}}$, etc. All these equations closely represents Pythagorean theorem. The program took around 1 hour and 32 minute for 10 runs.

NB. Detailed output mentioned in section 4.1, also can be found in github repository.

3.1 Limitation

After 500 generation the latest population observed to have couple of same individual. 4 individuals of the population have fitness value of 0. 8201, 149 are having 0. 8929, 87 are having 0.8900 and 96 are having 17.6507. The tree structure of individuals, who are of same fitness, are same. This phenomenon could have been avoided by reducing crossover mutation rate but doing so the chances of getting result significantly reduced. As mentioned earlier choosing primitive set and toolbox component is import. There may be some other combination of crossover, mutation or selection method that deals perfectly with the same.

The power function only takes rational exponents in form of p/q (where p, q both are integer). Doing this we are omitting some possible subtree like A^B or $B^{\frac{A}{B}}$ etc. Also as we are taking rational exponents there is no way to take irrational numbers in the power function. Maybe with the use of other crossover and mutation we can remove this limitation over power function.

Terminal only have value from range 1 to 10. First, we started with terminal of range [1,4] and when it was a success, we gradually moved up to 10. But 10 is not any special number! We can move on to any integer in practice. The only limitation is the computational power. More the computational power, more large number we can allow to throw off. So theoretically there is no limitation on terminal value.

4 Appendix

The algorithm gives output as string representation of tree like $power(add(mul(A,A),mul(B,B)),1,2)$ that can be expressed as $\sqrt{A^2+B^2}$. In fig 5.1 the same equation represented as tree.

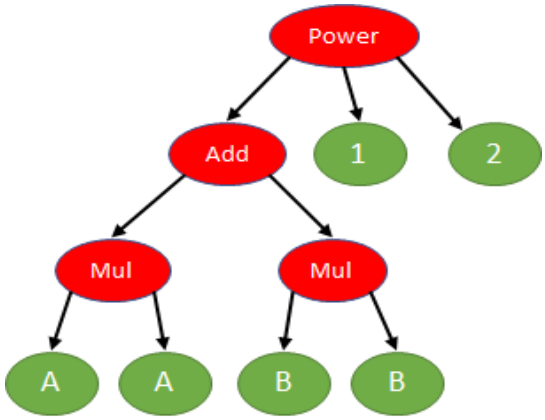

Fig 5.1: A tree structure of Pythagorean Equation

Not every time this algorithm gives the same result. Some non-trivial and apparently complicated trees which finally leads to the Pythagorean theorem are –

- `power(add(mul(A,A),mul(B,B)),1,2)`
- `power(add(mul(B,B),mul(A,A)),2,4)`
- `power(add(mul(B,B),mul(A,A)),3,6)`
- `power(add(mul(B,B),mul(A,A)),4,8)`
- `power(add(mul(A,A),mul(B,B)),5,10)`
- `power(add(mul(A,A),power(B,8,4)),1,2)`
- `power(add(power(A,8,4),mul(B,B)),2,4)`
- `power(add(power(A,4,2),mul(B,B)),5,10)`
- `power(add(sub(mul(A,A),sub(A,A)),mul(B,B)),1,2)`
- `power(add(sub(A,A),add(mul(A,A),mul(B,B))),1,2)`
- `power(add(mul(power(B,4,2),safe_div(A,A)),add(mul(A,A),sub(B,B))),4,8)`
- `power(sub(add(mul(A,A),sub(A,B)),sub(sub(A,B),mul(B,B))),5,10)`
- `power(add(mul(B,B),mul(A,power(power(A,4,3),3,4))),3,6)`
- `power(sub(mul(add(B,A),A),sub(mul(B,A),mul(B,B))),2,4)`
- `add(power(add(add(mul(B,B),sub(A,A)),mul(A,A)),2,4),safe_div(A,add(mul(B,B),mul(power(B,10,10),A))))`
- `power(add(add(mul(A,A),mul(B,B)),safe_div(add(B,A),power(B,5,1))),4,8)`

4.1 Run Result

We have achieved 8 successful runs out of 10 test runs. The result are as follows.

- `power(add(mul(A, A), mul(B, B)), 1, 2)`
- `power(add(mul(B, B), mul(A, A)), 3, 6)`
- `power(sub(mul(B, B), sub(safe_div(power(B, 3, 5), mul(B, B)), mul(A, A))), 4, 8)`
- `power(sub(add(mul(A, A), mul(B, B)), safe_div(sub(B, B), A)), 2, 4)`
- `power(add(mul(A, A), add(sub(B, B), mul(B, B))), 4, 8)`

- `power(mul(add(mul(A, A), power(B, 6, 3)), add(mul(A, A), power(B, 6, 3)) ), 2, 8)`
- `power(add(mul(A, A), mul(B, B)), 4, 8)`
- `power(add(mul(A, A), mul(B, B)), 5, 10)`

All there generated tree is having fitness value near 0 i.e. 0.0407. This error is due to choosing input point as float.
And the other 2 unsuccessful runs we got –

- `add(B, safe_div(add(power(A, 5, 6), mul(A, A)), add(safe_div(mul(A, A), add(A, B)), add(B, B))))`

which has fitness value of 3.795

- `add(B, safe_div(safe_div(add(mul(B, A), B), safe_div(B, A)), add(add(A, B), safe_div(mul(B, B), add(B, A)))))`

which has fitness value of 3.816

5 References

[1] Holland, J.H., Adaptation in natural and artificial systems1975, Ann Arbor: University of Michigan Press.
[2] Gen, M. and R. Cheng, Genetic algorithms and engineering design 1997, New York John Wiley & Sons.
[3] Holland's schema theorem
[4] Human-competitive results produced by genetic programming by John R Koza
[5] G. Smits, M. Kotanchek, "Pareto-front exploitation in symbolic regression" in Genetic Programming Theory and Practice II, MI, Ann Arbor: Springer-Verlag, pp. 283-299, May 2004.
[6] Martí, R., M.G.C. Resende, and C.C. Ribeiro, Multi-start methods for combinatorial optimization. European Journal of Operational Research, 2013. 226(1): p. 1-8.
[7] Son Duy Dao, Kazem Abhary, and Romeo Marian, An Adaptive Restarting Genetic Algorithm for Global Optimization
[8] Order of Nonlinearity as a Complexity Measure for Models Generated by Symbolic Regression via Pareto Genetic Programming by Ekaterina J. Vladislavleva, Guido F. Smits, Dick den Hertog. IEEE Transactions on Evolutionary Computation, vol. 13, no. 2, April 2009, p-334
[9] Kessaci, Y., et al., Parallel evolutionary algorithms for energy aware scheduling, in Intelligent Decision Systems in Large-Scale Distributed Environments, P. Bouvry, H. González-Vélez, and J. Kołodziej, Editors. 2011, Springer Berlin Heidelberg. p. 75-100.
[10] Dao, S.D., K. Abhary, and R. Marian, Optimisation of partner selection and collaborative transportation scheduling in Virtual Enterprises using GA. Expert Systems with Applications, 2014. 41(15): p. 6701-6717.
[11] Code Bloat Problem in Genetic Programming by Anuradha Purohit, Narendra S. Choudhari, Aruna Tiwari
[12] Preliminary Study of Bloat in Genetic Programming with Behaviour-Based Search by Leonardo Trujillo, Enrique Naredo and Yuliana Martínez
[13] Genetic Programming: On the Programming of Computers by Means of Natural Selection by JR Koza, Chapter 6, p114
[14] Operator Equalisation and Bloat Free GP by Stephen Dignum and Riccardo Poli

6 Conclusion and future work

Genetic Programming (GP) has ability to recognize the pattern directly from input points. GP is only the only tool for regression where the domain of the problem is unknown. After little tuning in the ‘Toolbox’ we got a remarkable result. We have tested our algorithm for Kepler's Law, the problem for which GP is famous for. It has been a success also.

To improve the performance of GP to reach the model, one should try different Selection, Crossover and Mutation methods. Also, for improvement one should using different approach on optimizing optima. There is more complex problem out there and many of them are unsolved. As a next step we would like to solve those problem to explore the aspects of regression via Symbolic Regression.