An Approach to Geometric Modelling Using Genetic Programming

[1]Snigdhajyoti Ghosh, [2]Aniket Bhunia, [3]Jyotirmay Karmakar,[4]Chira Ranjan Dutta, [5]Damodor Goswami

[1]Graduate Student, [2]Graduate Student, [3]Graduate Student, [4]Professor,[5]Professor

[1]Electronics And Communication Engineering,

[1]Netaji Subhash Engineering College, Kolkata, West Bengal, India-700152

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

Index Terms - Genetic Programming, DEAP, Genetic Algorithm, Pythagorean Triple.

I. 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. 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. 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'. 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

1.1 Genetic Algorithm and related works

Genetic Algorithm (GA) is a bio-inspired optimization technique, first introduced by Holland [1,3] and widely used in engineering design [2]. The algorithm first creates a random population, evaluate its fitness for selection and uses two genetic operators' 'crossover' and 'mutation' on those selected individuals to generate a new population.

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. GP has been successfully used as an automatic programming tool, a machine learning tool and an automatic problem-solving engine. John R. Koza mentions 76 instances where Genetic Programming has been able to produce results that are competitive with human produced results [4]..

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 without any prior assumption on the model structure. Genetic Programming is widely used for this purpose [5,6].

II. 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 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 are to match $f(a, b)$ with respective 'c'.

2.2 Creating Primitive Set

Primitive set refers to a set of operators used to construct the output function. So a set of basic math operators (Addition, Subtraction, Multiplication, Division, etc.) can be used as the primitive set. The leaf or terminal node 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 ‘Zero Division Error’ and Power for taking rational exponents. And added a ‘Terminal’ of multiple value i.e. called ‘Ephemeral Constant’.

2.3 Preparation of Toolbox and Statistics

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 rules and set of parameters. 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 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. To evaluate fitness, we used mean square error (MSE). For statistical calculations we used the module ‘NumPy’.

2.4 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.

2.5 Extension of DEAP Software

We needed to modify the software to modulate the stopping criteria. Sometimes it becomes difficult to achieve global optima with simple GP algorithm. 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].

III. Experimentation

We started with 50 data points but the success rate was very low. It was taking too much time to reach the optima and most of the time it converged to local optima instead of finding the global one. The probability of jumping out from local optima was low because the number of input points was not enough in number.

So, then we moved 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. 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 nodes in the tree representation) of generated individuals.

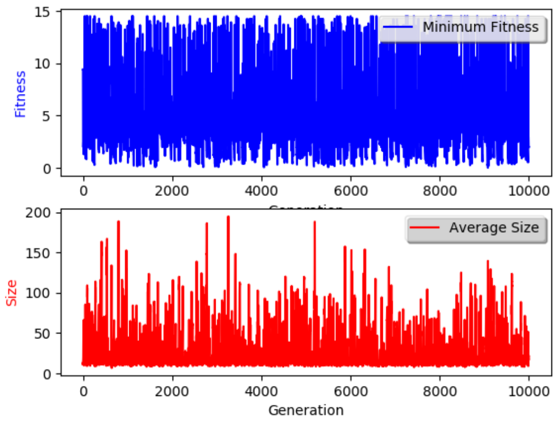

Fig 1

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 larger tree depth leads 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 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 hand, in fig 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 which allows to get simples models.

We fixed a resetting point after 20 generation. As we observed the maximum time the fitness stuck around 10th generation (In fig 3 the model converges after 10th gen).

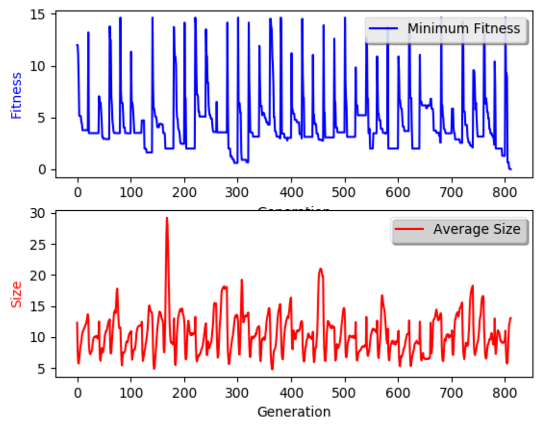

Fig 2

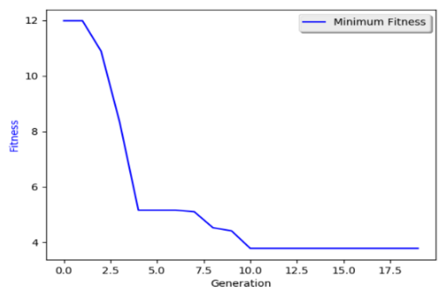

Fig 3: Generation vs Fitness

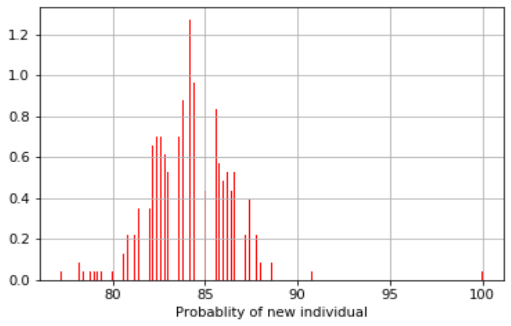

Fig 4: Effects of selection Crossover and Mutation

So, every after 20th generation the population get reset. With that we got success in finding solution. In fig 4 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. Uniform mutation happens at rate of 20%. As a result, almost 85% of the population changes in the next generation.

IV. RESULTS AND DISCUSSION

4.1 Results of Descriptive Statics of Study

With this approach, we have achieved 8 successful runs from 10 runs. And got results like $c = \sqrt{A^2 + B^2}$, $c = \sqrt{A^2 + B^2 - B^{-7/5}}$, $c = \sqrt{A^2 + B^2 + \frac{B-B}{A}}$, etc. All of these equations closely represent the Pythagorean theorem. The program took around 1 hour and 32 minutes for 10 runs. Detailed output mentioned in section 4.1, also can be found in GitHub repository.

4.2 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 the same equation represented as tree. 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)

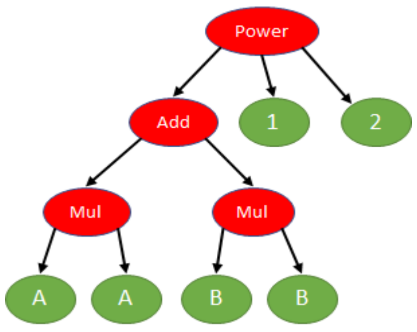

Fig 5: A typical tree structure of Pythagorean Equation

4.3 Run results

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

i) power(add(mul(A, A), mul(B, B)), 1, 2)
ii) power(add(mul(B, B), mul(A, A)), 3, 6)
iii) power(sub(mul(B, B), sub(safe_div(power(B, 3, 5), mul(B, B)), mul(A, A))), 4, 8)
iv) power(sub(add(mul(A, A), mul(B, B)), safe_div(sub(B, B), A)), 2, 4)
v) power(add(mul(A, A), add(sub(B, B), mul(B, B))), 4, 8)
vi) power(mul(add(mul(A, A), power(B, 6, 3)), add(mul(A, A), power(B, 6, 3))), 2, 8)
vii) power(add(mul(A, A), mul(B, B)), 4, 8)
viii) power(add(mul(A, A), mul(B, B)), 5, 10)

All these generated trees are having fitness value near 0 i.e. 0.0407. This small error is due to choosing input points as floating number. And the other 2 unsuccessful runs are:

i) 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

ii) 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

.

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