Genetic Programming for Soft Computing

[1]K.Thyagarajan, [2]R. Chinnaiyan

Abstract

This paper is dealt with the basics of genetic programming. The Genetic Programming (GP) is an important tool to evolve computer programs. An automatic programming has been the goal of scientists and engineers for a number of decades. Scientists would like to give the computer a problem and ask the computer to build a program to solve it. Genetic programming shows the most potential ways that automatically write computer programs for solving complex problems in computer science. Genetic Programming plays a vital role in future soft computing since it gives consistent solutions

Keywords : Automatic Programming, Genetic Programming, Soft Computing

1. Introduction

John Koza is the originator of the field of genetic programming. He started writing papers on genetic programming starting in 1989 (but he first applied for a patent in 1988). It turns out that others had the idea of evolving programs represented as trees before John Koza, but this work had gone relatively unnoticed. John Koza [6] demonstrated that genetic programming worked on a large number of Artificial Intelligence type problems and published a lot of papers GP is a powerful tool for automatically evolving computer programs. In general, the program evolved by GP can produce the same solution humans use to solve the target problem, or something completely new, perhaps better than the "conventional" manually designed solution.

"Genetic programming is an automated method for creating a working computer program from a high-level problem statement of a problem. Genetic programming does this by genetically breeding a population of computer programs using the principles of Darwinian natural selection and biologically inspired operations. " Genetic programming is modification of genetic algorithms with one major difference. The population consists of individuals represented by specific data structure – trees. Genetic programming iteratively transforms a population of computer programs into a new generation of programs by applying analogs of naturally occurring genetic operations.

Genetic programming is a search technique that explores the space of computer programs. The search for solutions to a problem starts from a group of points in this search space. Those points are then used to generate a new generation of points through crossover, mutation, and reproduction and possibly other genetic operations. This process is repeated over and over again until a termination criterion is satisfied.

1.1 Genetic Algorithm (GA) vs Genetic Programming (GP)

The GP technique is an evolutionary algorithm that bears a strong resemblance to genetic algorithm's (GA's). The

[1]HOD, Department of Computer Science, AVC College (Autonomous), Mannampandal – Mayiladuthurai, Nagappattinam District – 609 305.

[2]Senior Lecturer, Department of Computer Applications, A.V.C College of Engineering, Mannampandal – Mayiladuthurai, Nagappattinam District – 609 305.

primary differences between GA's and GP can be summarized as follows;

(i) GP typically codes solutions as tree structured variable length chromosomes, while GA's generally make use of chromosomes of fixed length and structure.

(ii) GP typically incorporates a domain specific syntax that governs acceptable (or meaningful) arrangements of information on the chromosome. For GA's, the chromosomes are typically syntax free.

(iii) GP makes use of genetic operators that preserve the syntax of its tree-structured chromosomes during 'reproduction'.

(iv) GP solutions are often coded in a manner that allows the chromosomes to be executed directly using an appropriate interpreter. GA's are rarely coded in a directly executable form.

2. Basic Genetic Operations

Having applied the fitness test to all the individuals in the initial random population, the evolutionary process starts. Individuals in the new population are formed by two main methods: reproduction and crossover. Once the new population is complete (i.e. the same size as the old) the old population is destroyed. Each 'individual' in a generation represents, with its chromosome, a feasible solution to the problem; in our case, a discriminate function to be evaluated by a fitness function. The best individuals are continuously being selected, and crossover and mutation take place. Following a number of generations, the population converges to the solution that best represents the discrimination function (Figure. 1).

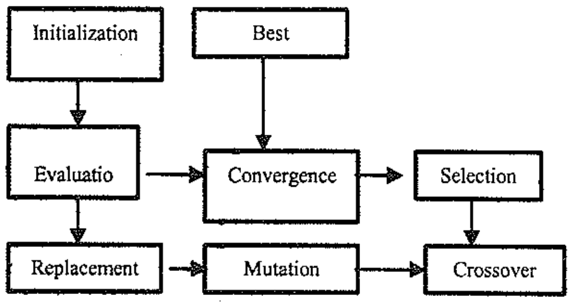

Figure 1: Block Diagram of Genetic Programming

2.1 Reproduction

An asexual method, reproduction is where a selected individual copies itself into the new population. It is effectively the same as one individual surviving into the next generation. Koza [6] allowed 10% of the population to reproduce. If the fitness test does not change, reproduction can have a significant effect on the total time required for GP because a reproduced individual will have an identical fitness score to that of its parent. Thus a reproduced individual does not need to be tested, as the result is already known. For Koza [6], this represented a 10% reduction in the required time to fitness test a population. However, a fitness test that has a random component, which is effectively a test that does not initialize to exactly the same starting scenario, would not apply for this increase in efficiency. The selection of an individual to undergo reproduction is the responsibility of the selection function.

2.2 Crossover

Organisms' sexual reproduction is the analogy for crossover. Crossover requires two individuals and produces two different individuals for the new population. In this technique genetic material from two individuals is mixed to form off- spring. Koza [6] uses crossover on 90% of the population—it is the more important of the two methods because it provides the source of new (and eventually better) individuals. There are a few other evolutionary operations: editing, mutation, permutation, encapsulation and decimation, most of which were ignored by Koza in his earlier work but have recently been given more consideration

3. Steps in Genetic Programming

Genetic algorithms create a string of numbers that represent the solution. Genetic Programming uses four steps (Figure. 2) to solve problems:

1) Generate an initial population of random compositions of the functions and terminals of the problem (computer programs).
2) Execute each program in the population and assign it a fitness value according to how well it solves the problem.
3) Create a new population of computer programs.
   i) Copy the best existing programs
   ii) Create new computer programs by mutation.
   iii) Create new computer programs by crossover
4) The best computer program that appeared in any generation, the best-so-far solution is designated as the result of genetic programming

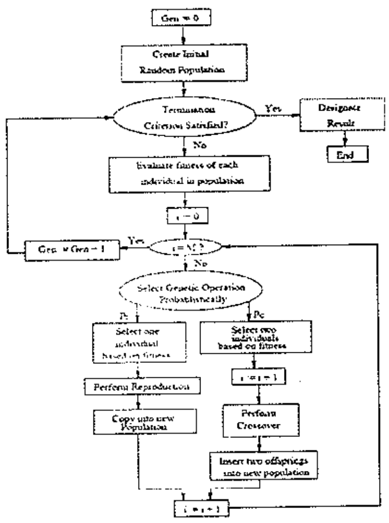
Gen = 0
Create Initial Random Population
Termination Criterion Satisfied?
Yes
Designate Result
End
No
Evaluate fitness of each individual in population
i = 0
Gen = Gen + 1
Yes
i = M ?
No
Select Genetic Operation Probabilistically
Pr
Pc
Select one individual based on fitness
Select two individuals based on fitness
Perform Reproduction
i = i + 1
Copy into new Population
Perform Crossover
Insert two offsprings into new population
i = i + 1

Figure 2: Steps in Genetic Programming

4. Applications of GP

Genetic Programming can be applied to the following areas where you simply have no idea how to program a solution, but where you know what you want a) control, b) bioinformatics c) classification, d) data mining, e) system identification, f) forecasting, g) satellite image data, h) astronomical data, i) petroleum databases etc.,

5. Implementation of Traveling Salesman Problem (TSP) using GP

A method of solving the Traveling Salesman Problem (TSP) using Genetic Algorithms [13] has been developed. In the TSP, the goal is to find the shortest distance between N different cities. The path that the salesman takes is called a tour. Testing every possibility for an N city tour would be N! math additions. A 30-city tour would be 2.65 * 1032 additions. Assuming 1 billion additions per second, this would take over 8,000,000,000,000,000 years. Adding one more city would cause the number of additions to increase by a factor of 31. Obviously, this is an impossible solution. A genetic algorithm can be used to find a solution is much less time. Although it probably will not find the best solution, it can find a near perfect solution in less than a minute. There are two basic steps for solving the traveling salesman problem using a GA (and so would be in a searching for optimal Petri net). First, create a group of many random tours in what is called a population. These tours are stored as a sequence of numbers.

Second, pick 2 of the better (shorter) tours parents in the population and combine them, using crossover, to create 2 new solutions children in the hope that they create an even better solution. Crossover is performed by picking a random point in the parent's sequences and switching every number in the sequence after that point. The idea of Genetic Algorithms is to simulate the way nature uses evolution. The GA uses Survival of the Fittest with the different solutions in the population. The good solutions reproduce to form new and hopefully better solutions in the population, while the bad solutions are removed. Eventually, the GA will make every solution look identical. This is not ideal. There are two ways around this. The first is to use a very large initial population so that it takes the GA longer to make all of the solutions the same. The second method is mutation. Mutation is when the GA randomly changes one of the solutions. Sometimes a mutation can lead to a better solution that a crossover would not have found. The difficulty in the TSP using a GA is encoding the solutions [13]. Similar problem seems to occur in Petri nets. Terminating function should include a forward accessibility condition and a natural representation is advisable.

6. Conclusion

This survey paper has revealed the concept of genetic programming and the GP has much to offer for building fast, domain specific systems, a welcome experimental orientation, and anytime behavior during evolution. It also has many shortcomings: it is CPU-intensive, it is sensitive to minor changes in parameters, and it does not yet reliably produce robust results.

The constant technological improvements in genetic programming technology and its concrete foundations with computing power, genetic programming has been able to solve tens of complicated problems with human-competitive results in the recent past. In a few years' time genetic programming will be able to routinely and competently solve important problems for us in a variety of specific domains of application, even when running on a single personal computer, thereby becoming an essential collaborator for many human activities

7. References

1. Alba .E, Cotta .C and Troyo .J.J, *"Type constrained genetic programming for rule based definition in fuzzy logic controllers"*, Proc. First Annual. Conf. on Genetic Programming - GP'96, Stanford University, USA, PP 255 - 260, 1996.
2. Bettenhausen .K.D, Marenbach .P, Freyer .S, Rettenmaier .H and Nieken .U, *"Selforganising structured modelling of a biotechnological fed-batch fermentation by means of genetic programming"*, Proc. IEE Conf. on Genetic Algorithms in Engng. Systems: Innovations and Applications - GALESIA'95, Sheffield, UK, No.414, PP 481-486, 1995.
3. Cramer .N.L, *"A representation for the adaptive generation of simple sequential programs"*, Proc. Int. Conf. on Genetic Algorithms and their Applications, Carnegia Mellon University, Pittsburgh, USA, PP 183-187, 1985.
4. Hampo. R.J, *"Genetic programming: A new paradigm for control and analysis"*, Proc. 3rd ASME Symposium on Transportation Systems, Anaheim, USA, PP 155-163, 1992 Handly. S, "The automatic generation of plans for a mobile robot via genetic programming with automatically defined functions", Kinnear, K.E. (Ed.), *"Advances in Genetic Programming*, 18, PP 391-407, 1994.
5. Koza .J.R and Keane .M.A, *"Genetic breeding of non-linear optimal control strategies for broom balancing"*, Proc. 9th Int. Conf. on Analysis and Optimization of Systems, France, PP 47-56, 1990.
6. Koza .J.R, *"Genetic Programming : On the Programming of Computers by Means of Natural Selection"*, The MIT Press, 1992.

7. Koza .J, *"Genetic programming: On the programming of computers by means of natural selection"*, The MIT Press, USA, 1992 Koza .J, *"Genetic Programming II: Automatic Discovery of Reusable Programs"*, The MIT Press, USA, 1994.
8. Koza .J. R, *"Genetic Programming II: Automatic Discovery of Reusable Programs"*, The MIT Press, 1994.
9. Koza .J.R, Bennett .F.H, Andre .D and Keane .M.A, *"Automated WYWIWYG design of both the topology and component values of electrical circuits using genetic programming"*, Proc. of the First Annual Conf. on Genetic Programming - GP'96, Stanford University, USA, PP 123-131, 1996.
10. Koza .J.R, Andre .D, Bennett .F.H and Keane .M.A, *"Use of automatically defined functions and architecture altering operations in automated circuit synthesis with genetic programming"*, Proc. of the First Annual Conf. on Genetic Programming - GP'96, Stanford University, USA, PP 132-140, 1996.
11. Koza .J.R, *"Future Work and Practical Applications of Genetic Programming"*, In Handbook of Evolutionary Computation, PP H1.1:3. IOP Publishing Ltd and Oxford University Press, 1997.
12. Kang .S.J, Jang .S.H, Hwang. H.S, Woo, K.B ; *"Colored timed Petri nets modeling and job scheduling using GA of semiconductor manufacturing"* In: IEICE Trans. on Information and Systems, Vol. E82-D, No. 11, PP 1483-1485.
13. LaLena .M, *"Travelling Salesman Problem Using Genetic Algorithms"* http://www.lalena.com/ai/tsp/
14. Takahashi .K, Yamamura .M, Kobayashi .S, *"A GA approach to solving reachability problems for Petri nets"*, In: IEICE Trans. on Fundamentals in Electronics, Communications and Computer Science, Vol. E79-A, No. 11, PP 1774-1780, 1996.

Author's Biography

K.Thyagarajan is currently working as a Lecturer (SG) and Head, Department of Computer Science, A.V.C. College (Autonomous), Mayiladuthurai. He had over 21 years of teaching experience. His areas of interest include Data Mining, Image Processing and Algorithms. He had received M.Phil Computer Science Degree and M.Phil Mathematics Degree from Bharathidasan University. Currently he is doing Ph.D in Computer Science in Bharathidasan University.

R.Chinnaiyan is currently working as a Senior Lecturer in the Department of Computer Applications of A.V.C. College of Engineering, Mayiladuthurai. He had over 7 years of teaching experience. His areas of interest include Software Systems Reliability, OOAD and Data Mining. Now he is doing Ph.D in Anna University at CIT, Coimbatore.