IOSR Journal of Computer Engineering (IOSR-JCE)
e-ISSN: 2278-0661,p-ISSN: 2278-8727
PP 10-15
www.iosrjournals.org

Automated Code Generator

Mahenaz Sayyed[1], NamrataNathani[2],Yash Chugh[3], Seema Kolkur[4]
[1]*(Department of Computer Science, TSEC, Mumbai University, India)*
[2]*(Department of Computer Science, TSEC, Mumbai University, India)*
[3]*(Department of Computer Science, TSEC, Mumbai University, India)*
[4]*(Department of Computer Science, TSEC, Mumbai University, India)*

Abstract: *Is it possible for a computer program to writeitsown programs? Couldhuman software developersbereplaced one day by the very computers thatthey master? The objective of thispaperis to create an Artificial Intelligence system thatis capable of generating codes for a giventargetproblemstatementspecified by the user. The system makes use of GeneticAlgorithm. It startswith the gene set of all possible instructions of the targetlanguage and absolutely no knowledge of the syntax and semantics. Firstlyitgenerates an initial population of randomsequences of the instruction set. Fitness function of eachindividualiscalculated. The desired and obtained outputs are compared in the fitness function. The fitness functionis the basis for population evolution in future generations. The processterminateswhen the desired value of fitness score isachieved. This paperdemonstrates a prototype of the revolutionizing concept of employing machines to generate codes. Withadequate memory and processors, high speed and accuracycanbeachieved.*

***Keywords** -Intelligence; Genetic Algorithm; Fitness function; Fitness score.*

I. Introduction

The inspiration was derived from the infinite monkey theorem, which states if there exists 1,000+ monkeys banging on a typewriter for a period considerable enough, they will eventually re-produce a play written by Shakespeare. This is the idea that goes to makea Genetic Algorithm(GA).A GA is a type of artificial intelligence, modeled after biological evolution that begins with no knowledge of the subject, aside from available tools and valid instructions. Genetic algorithms are programmatic implementations of survival of the fittest.They can also be classified as artificially intelligent search algorithms, with regard to how they search an immense problem space for a specific solution.

II. Genetic Algorithm

A genetic algorithm (GA) is a method for solving both constrained and unconstrained optimization problems based on a natural selection process that mimics biological evolution. The algorithm repeatedly modifies a population of individual solutions. Fig.1 shows the flowchart of GA. Genetic Algorithms are the heuristic search and optimization techniques that mimic the process of natural evolution [1].

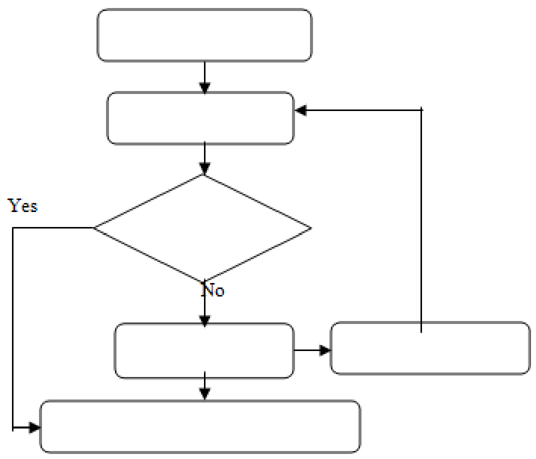


Fig. 1 flowchart of genetic algorithm

Some of the terminologies related to GA are explained below:

Chromosome: It contains information about solution that it represents. A chromosome is sub divided into genes [2]. Fig. 2 displays a chromosome comprised of genes encoded in binary format.

1 0 1 0 1 1 1 0 1 1 1 1 0 1 0 1

Fig. 2 chromosome

Gene: Genes are the basic "instructions" for building a generic algorithms. A chromosome is a sequence of genes. Genes may describe a possible solution to a problem, without actually being the solution [2]. As you can see in Fig.3, a chromosome is nothing but set of genes.

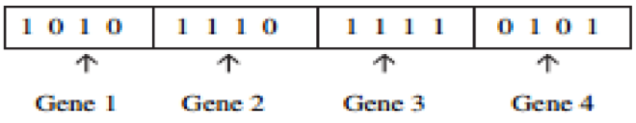


Fig. 3 genes representation of chromosome

The various steps involved in a genetic algorithm are as follows:-

2.1 Selection

This process determines which solutions are to be preserved and allowed to reproduce and which ones deserve to die out. The primary objective of the selection operator is to emphasize the good solutions and eliminate the bad solutions in a population while keeping the population size constant. 'Selects the best, discards the rest' [3].There are different techniques to implement selection in genetic algorithms.
They are:

- Tournament selection
- Roulette wheel selection
- Proportionate selection
- Rank selection
- Steady state selection

2.2 Encoding

The process of representing a solution in the form of a string that conveys the necessary information is called Encoding. Just as in a chromosome, each gene controls a particular characteristic of the individual, similarly, each bit in the string represents a characteristic of the solution. Most common method of encoding is binary encoding. Chromosomes are strings of 1 and 0 and each position in the chromosome represents a particular characteristic of the problem. Fig. 4 shows binary encoding of colours.
The System will be encoded using the Instruction Set of Brain F*ck language. Thus the Chromosome will appear as strings of operators in Brainfuck.

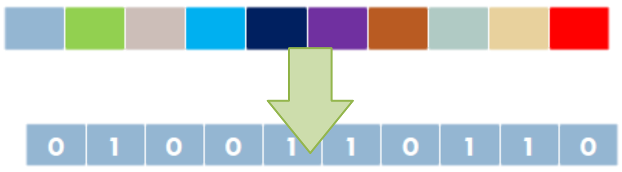


Fig. 4 binary encoding

2.3 Crossover

Crossover is used to create new solutions from the available solutions present in the mating pool after applying selection operator. This operator exchanges the gene information between the solutions in the mating pool [3].
The most common types of crossovers are:

2.3.1 *Single point crossover*

A single crossover point on both parents' organism strings is selected. All data beyond that point in either organism string is swapped between the two parent organisms as shown in Fig.5. The resulting organisms are treated as the child chromosomes.

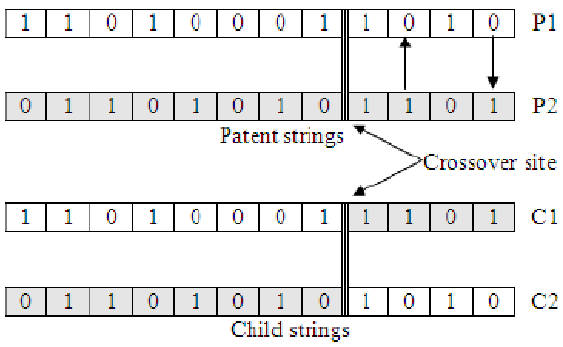


Fig. 5 crossover type-I

2.3.2 Two point crossover

Two-point crossover calls for two points to be selected on the parent organism strings which is explained in Fig. 6. Everything between the two points is swapped between the parent organisms, rendering two child organisms.

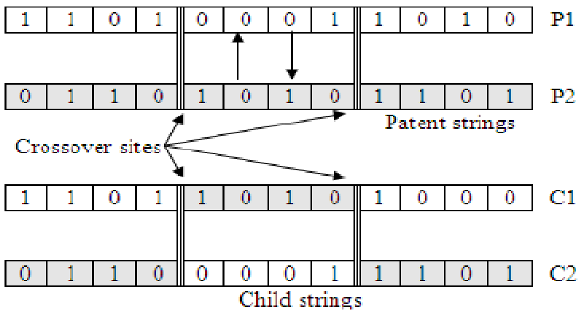


Fig. 6 Crossover type-II

2.4 ***Mutation***

Mutation is the occasional introduction of new features in to the solution strings of the population pool to maintain diversity in the population. Though crossover has the main responsibility to search for the optimal solution, mutation is also used for this purpose.

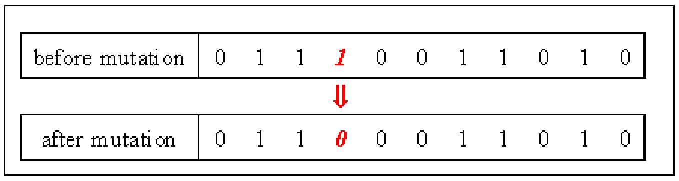


Fig. 7 Mutation

Fig. 7 demonstrates the process of mutation. A random bit is chosen (4th bit in this case) and is flipped. Mutation may or may not occur. After performing these operations a new generation is obtained. This is then looped until Target Fitness is achieved.

III. Existing Solution

3.1 Automatic python programming using stack based genetic programming

Existing systems prevalent include systems to evolve byte code of python programming language by stack based genetic programming. Python is flexible and popular programming language powered by plenty of research tools. The existing solution includes methods to produce successful python codes for two regression problems [3].

3.2Automatic programming using genetic programming

The paper focuses on the procedural/imperative programming paradigm. The generational GP algorithm was implemented using the grow method to create the initial population, tournament selection to choose parents and reproduction, crossover and mutation for regeneration purposes. The paper also presents a form of incremental learning which facilitates modularization. The GP approach to automatic programming was tested on ten programming problems that are usually presented to novice programmers in a first year procedural programming course of an undergraduate degree in computer science [4].

IV. Proposed System

The implementation of the system is done in a twostep fashion:-

1. Using JAVA, take the instruction set as input and apply genetic algorithm to produce output(A generation of programs). Feed this output to the target language.
2. Execute the output in the target language and feed this output back to the java program to calculate the fitness score of each individuals(programs) using a fitness function.

The target language used here is brainfuck. It is turing complete and has an instruction set of just eight instructions.Fig. 8 describes the flowchart of the system.

Instruction set: {{+} , {-} , {>} , {<} , {. } , { , } , { [} , {] }}

{+}: increment the byte at the data pointer by one.

{-}: decrement the byte at the data pointer by one.

{>}: increment the data pointer to next cell.

{<}: decrement the data pointer to next left cell.

{.}: output the byte at the data pointer.

{,}: accept one byte of input, storing its value in the byte at the data pointer.

{[}: if the byte at the data pointer is zero, then instead of moving the instruction pointer forward to the next command, jump it forward to the command after the matching] command.

{] }:if the byte at the data pointer is nonzero, then instead of moving the instruction pointer forward to the next command, jump it back to the command after the matching [command.

The Automated Code Generator works as follows:

1. A genome consists of an array of doubles.
2. Each gene corresponds to an instruction in brainf-ck programming language.
3. Start with a population of random genes.
4. Decode each genome into a resulting program by converting each double into its corresponding instruction and executing the program.
5. Get each programs fitness score, based upon the output it writes to the console(if any), and rank them.
6. Mate the best genomes together using roulette selection, crossover and mutation to produce a new generation.
7. Repeat the process with new generation until the target is achieved.

4.1 Encoding

One of the primary tasks in genetic algorithms is the appropriate encoding of the genes to facilitate efficient crossover and mutation to arrive at the solution. The Genomes for the system will be encoded using the Brainfuck instruction set.

Example: ++<<>>,.>>,.—

4.2 Fitness calculation

The basic criterion for each fitness function remains fundamental which is comparison between the desired output and the obtained output.

FITNESS = FITNESS + 256 – Math. Absolute (Desired – Obtained);

Where FITNESS is the current fitness of the chromosome and 256 is used for 256 ASCII characters.

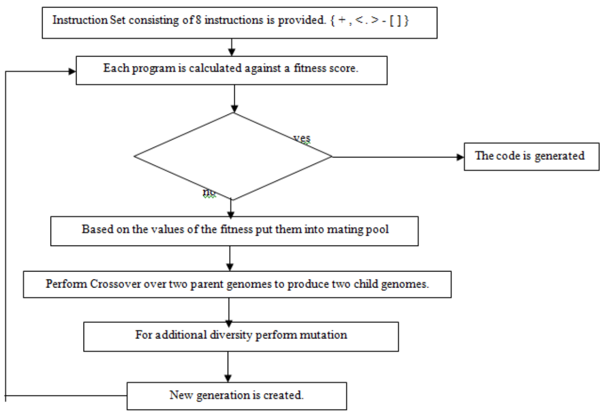

Instruction Set consisting of 8 instructions is provided. { + , < . > - [ ] }
Each program is calculated against a fitness score.
yes
The code is generated
no
Based on the values of the fitness put them into mating pool
Perform Crossover over two parent genomes to produce two child genomes.
For additional diversity perform mutation
New generation is created.

Fig. 8 system flowchart

4.3 Selection

The system uses fitness proportionate selection also known as roulette wheel selection. The basic part of the selection process is to stochastically select from one generation to create the basis of the next generation. The requirement is that the fittest individuals have a greater chance of survival than weaker ones. This replicates nature in that fitter individuals will tend to have a better probability of survival and will go forward to form the mating pool for the next generation. Weaker individuals are not without a chance[6].

Table 1 Selection Process

Chromosomes	Fitness Value	Probablity	Expected Count	Actual Count
++>>.+.+.><<<--	475	0.3247	6	2
--+++>.>+.<<--.-	25	0.0581	1	0
++>>.+>.>+.<<--	625	0.4011	7	3
>.+>.>.+.><++.<	210	0.1126	3	1

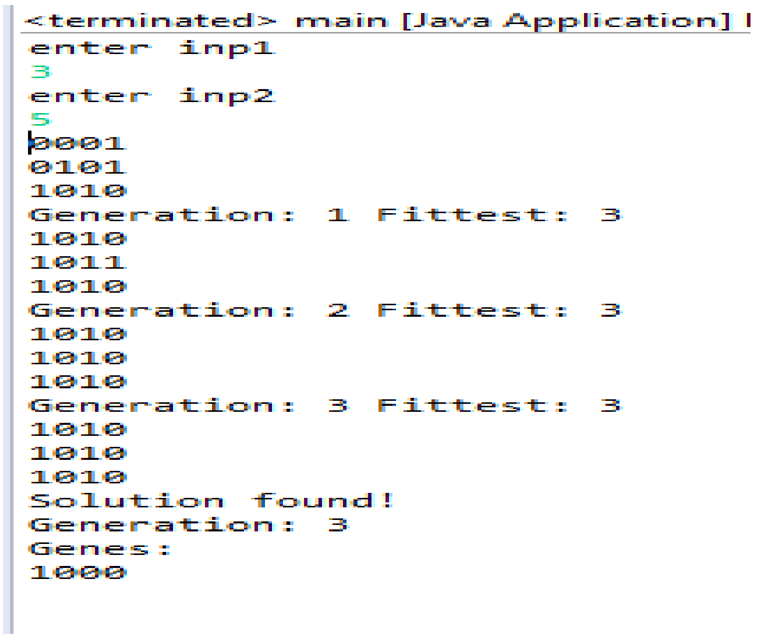
To demonstrate the working of the system a random population of 5 programs has been initialized and fitness value of each program is calculated for the selection process. The details are explained in table 1.

4.4 *Crossover*

The System will implement a single point crossover randomly chosen. This way a new generation is created and the entire cycle is repeated to achieve an approximate answer.

V. Implementation

Two modules were implemented. The first module accepted two numbers from the user and calculated the sum of the two numbers. The second module was given a string of characters and it was supposed to generate the target string.The instruction set {0,1} was taken as input.Fig. 9 shows the output of first module. Two numbers were taken as inputs from the user (inp1= 3 and inp2= 5). As it can be seen below, after 3 generations it was able to find the solution.



```
<terminated> main [Java Application] I
enter inp1
3
enter inp2
5
0001
0101
1010
Generation: 1 Fittest: 3
1010
1011
1010
Generation: 2 Fittest: 3
1010
1010
1010
Generation: 3 Fittest: 3
1010
1010
1010
Solution found!
Generation: 3
Genes:
1000
```

Fig. 9 Results

VI. Conclusion

Although the scope of the system now is restricted to basic logic gates and string manipulations this idea however wants to revolutionize the process of code generation. The current system can be used a s a background process to achieve the code while the user performs other tasks. It aims to automate the working of machines to save human time and effort. If implemented in an efficient manner it also optimizes the codes. With the use of super computers and extremely fast processors the generation time will be reduced exponentially.

References

[1] R.K. Bhattacharjya/CE/IITG," Introduction To Genetic Algorithms"

[2] S.N.Sivanandam, S.N.Deepa "Principles of Soft Computing" Second Edition, Wiley Publication..

[3] Hyun Soo Park, kyungJoong Kim "Automatic python programming using stack based genetic programming",GECCO'12 Companion, July 7-11,2012, Philadelphia, PA, USA. ACM 978-1-4503-1178-6/12/07.

[4] Kevin Igwe and Nelishia Pillay, "Automatic Programming Using Genetic Programming", 2013 Third World Congress on Information and Communication Technologies (WICT).

[5] Stuart J. Russell and Peter Norvig, "Artificial Intelligence A Modern Approach " Second Edition , Pearson Education.