An Algorithm for Finding the Optimal Path In Basis Path Testing using GABVIE Model

Seema Sharma, Shaveta Bhatia

Abstract*: Software testing is one of the most vital factors in software development life cycle. It is mainly used for testing the program code, known as white box testing and to test the functionality of the program, known as black box testing. Manual generation of test data is very costly, error vulnerable and time consuming task. Subsequently, there is a need to make the process automated as could be expected under the circumstances. This paper presented the automated generation of optimal path with intention of attaining the maximum coverage. The work being done considers the optimal path coverage in minimum cost. The task of generating test cases can be done through the concept Genetic Algorithm with importance of variable (GABVIE). The proposed algorithm additionally considers programs having numerous modules. This is vital as a large portion of the current test data generators not succeeded to establish the communication between the modules. The approach has been implemented on various program code and the outcomes got have been confirmed. The proposed work considers the white box testing.

Keywords: Variables, test cases, White box testing, Genetic Algorithm

I. INTRODUCTION

Testing is the methodology of running a program code with the aim of discovering errors [1]. It is a process for evaluating the software, to evaluate the functionality of a software application with an intention to identify the bugs present in the software and produce a defect free best quality product to the customer. Whatever methodology was applied on the software to test it should be operable and observable so that defects can be detect easily and changes in the software can be controlled by the tester. According to the present scenario, because of consistent change and improvement in digitization, our lives are improving in all regions. The manner in which we work is additionally changed. We get to our bank on the web, we do shopping on the web, many more. We are totally depending on digitized world and frameworks. Imagine a scenario where these frameworks turnout to be deficient [2]. One little bug in the system can have a tremendous effect on business .Therefore for delivery a good quality product; we need Software Testing in the Software Development Process.

Revised Manuscript Received on January 30, 2020.
***** Correspondence Author

Seema Sharma*, FCA, Manav Rachna international Institute of Research and Studies, Faridabad, India.
Dr. Shaveta Bhatia, FCA, Manav Rachna international Institute of Research and Studies, Faridabad, India.

As know that that “there must be a fixed time, and an appropriate environment for every activity done on earth”. But still the above phrase is as yet not utilized in Code testing. The Software testing still needs 'the perfect time' and 'the ideal condition.' It might be noticed that, 'the correct time' and 'the positive condition' are a result of best test case generation [3]. Manual test data generation is laborious and it takes significant level of time and cost. So, for solving such problem facing in manual testing the Automatic test case Generation should be progressively effective for handle time, cost and coverage. This will upgrade the unwavering quality of testing and the product

After the extensive literature review, we came to know that the one of the objective of best test cases is to maximize code coverage in significantly less time and minimum cost [4]. However, cost factor includes cost of executing the test suite and the cost of checking the system performance [10].

The work being done considers the maximum coverage and minimization of the cost [5]. This bi-target improvement benchmark can be accomplished by stepping through to generate the test cases for every individual way [6]. The task of generating test cases can be done through the concept Genetic Algorithm with importance of variable (GABVIE).

The model was proposed for finding out the most important variable to facilitate test case generation. The proposed algorithm was novel and has been implemented [8]. This paper discusses the implementation of the model and the observation therein. It may state the Genetic Algorithm has been used for the purpose of optimization. The results are promising. This works paves the way of the use of most important variable in generic test cases.

The paper has been organized as follows:

Section 1 is the literature review, Section II is the proposed model, Section III is implementation Section IV is Results, and Section V is Conclusion.

II. LITERATURE REVIEW

A broad survey has been piloted to examine the proposed methodology till now and discover the gaps in the existing works .A new strategy has been proposed on the basis of existing work. Ramamoorthy et al[29] and miller and spooner[37] have been developed the automated test data generation.The objectives behind their work was , evaluating program code, which have comprised in the program flow graph , to design a constraint generator and a test data generator. In the paper, Ramamoorthy [29] have generates the test cases through all three objectives constraint generator program flow graph .

A symbol table is also implemented which plays a fundamental role in generating test cases.

The problem faced in the above technique is the index used in array is again the array. The technique was tested for about 1000 FORTRAN programs which consist of more than 30 statements, and the result the time taken for processing is half per constraints.

The maximum time was consumed in backtracking. The technique works in the following way: the inputs were arranged in sequence and the constraints found ,were converted in Conjunctive normal form .The equations used in the technique were derived from the help of constraints and these equations were used in generating the test data.

Test cases have been generated by Miller and Spooner[37] used the numerical maximization technique rather conception rather than symbolic implementation of given path . The concept is used for numerical based programs or the programs that consists floating point data [21].

In 1976 Clarke[4] had developed an algorithm for automatic generation of test data .The concept was based on the symbolic execution (SE) of the path of given program code that defines a set of conditions of the input variables of source code. The symbolic execution uses depiction of path and useful data of program unit. The set of conditions were solved by an inequality solver and thus generates test data. The Programs of ANSI Fortran were taken as a experiments [4].

Different Techniques have been developed for producing automatic generation of Test Data where numerous strategies depend on heuristic approach.

In 2004 a extensive survey was conducted by McMinn[19],which was test case generation on the basis of search. It was found that various papers have used Genetic Algorithms (GAs) to solve the problem of Automatic test data generation [36, 27, 23, 35, 15, 28, 20, 29, 32, 11, 12, 31].

Xanthakis et al. in 1992 proposed the algorithm which was based on variable length genetic algorithm for optimization. The software path cluster was introduced and executes the most critical path first.

As the main objective of software testing is to achieve the maximum coverage of program source code. However it is not feasible for generating and executing the test cases that will appropriate for all the classes of source code. Therefore there is need to select the subset of classes that have to be tested. Giovanni Grano et al. (2018), investigates that tool of test data generation for achieving the coverage can be predicted by use source-code metrics. For attaining the above issue four different kinds of source-code features have been used. The experiments have been carried on more than 3000 java classes. Different machine learning algorithms were also compared and analyzed for investigating the most impact factors for prediction accuracy. The result shows that if the model achieves average 0.15 and .21 will be considered as best model.

Davi Silva Rodrigues et al. (2018) applies the genetic algorithm for generating test data on images, sounds, and on any complex 3D (three-dimensional) models. Since GA is non- deterministic by nature therefore it enables to investigate the hefty solution. The evaluation metric that would be used for evaluating these intricate data was execution time and quality assessment.

Miller et al. [23] used Genetic Algorithm and program graph for the development of proposed methodology. This technique was basically works on the criteria of branch coverage [23]. The name of the framework was TDGen, that works for Boolean data type variables and enumerated data types variables. This technique correlated with GADGET [38] with the various parameter, such as, coverage of path, Statement, Branch, Condition etc. Various experiments have been conducted on different standard programs and then comparing it with the existing tool GADGET[38].

III. SOFTWARE TESTING

Software testing: "Testing is the process of running a program with the aim of finding errors."[1].is a process for evaluating the software, to evaluate the functionality of a software application with an intention to identify the bugs present in the software and produce a defect free best quality product to the customer. Whatever methodology was applied on the software to test it should be operable and observable so that defects can be detect easily and changes in the software can be controlled by the tester. According to the present scenario, because of consistent change and improvement in digitization, our lives are improving in all regions. The manner in which we work is additionally changed. We get to our bank on the web, we do shopping on the web, many more. We are totally depending on digitized world and frameworks. Imagine a scenario where these frameworks turnout to be deficient. One little bug in the system can have a tremendous effect on business .Therefore for delivery a good quality product; we need Software Testing in the Software Development Process. Basically there are two ways through which the testing possible one is manual and other is automation testing. However in spite of availability of many techniques for software testing but mainly used, are functional (black box) and structural (white box) testing and Grey box testing. Black box testing focuses on functional needs of the software. It tests the software which is based only on the specification. Sometimes it becomes advantageous as the program structure code is not known to the tester.

In structural testing commonly known as white box testing, the structure of the program is known to the tester .The tester analyze the code and generates the test data for testing the program .Structural testing is focuses on pseudo-code of software. There are many methodology that have been used testing the source code like control flow testing, Data Flow testing, mutation testing etc. In Control Flow graph based testing the source code is represented by a directed graph whose vertices and edges signifies the program elements [15, 17]. There are certain coverage criteria like statement, branch, condition, and path and we may like to cover the reasonable coverage. The Data flow based testing is done by converting the control flow graph to optimized graph based on the usage and definition of the variable defined in the source code. Mutation testing is carried out by inserting the small mutants in the program.

Now the test case has been generated and can apply to the modified program and as well to the original program. If there will be significant change in both the outputs that means of successful completion .Later on the mutant is said killed [18, 19]

IV. GENETIC ALGORITHM

Since in evolutionary computation, a family of algorithms has been implemented in realistic problems and genetic algorithm (GA) is one of powerful algorithm in the entire family of evolutionary algorithm. It is a feasible, vigorous, heuristic approach and search methodology. The term GA (Goldberg 2006) is a heuristic approach works on the principle of, "nature evolves species of life and genetic selection of fittest individuals". In this approach ,the initial inputs taken as for solving the problem are presented by a population (called chromosomes) which is again goes through the operators of GA i.e cross over and mutation. . The initial population (i.e chromosome) may be taken as string of binary digits or real numbers. Every digit in this population is called a gene. The selection of initial population is chosen randomly or can be created manually. The steps of GA are as follows:

```
Step 1.Initialization of population (i.e initial random inputs will be taken called series of chromosomes)
Step 2:do
{
Step 2.Evaluation (each initial population will be evaluated through appropriate fitness function)
Step 3. Selection (selection of population according to fitness function)
Step 4.Apply
{
        a)Crossover (population)
        b)Mutate (population)
        c)Evaluate (population)
}
} While (maximum generation found || maximum coverage of path found)
```

Genetic Algorithm (GA) comprises of three operators Selection, Crossover and mutation which work on primary population:-

V. PROPOSED MODEL

The pivot concept behind the Genetic Algorithm based variable Importance evaluation (GABVIE) model is taking "variables" used in the program code as a constraint used to generate the test cases to test the path testing of a program code. The proposed algorithm assigns fitness to a variable depending upon its occurrence in the code and the frequency. The frequency of a variable is counted as follows. The occurrence of a variable in the 'if' block, assigns 1 to the frequency and that inside a 'for' loop (or 'while' for that matter) assigns n to it. The weight is determined by the frequency of the selected variables, multiplied by a constant λ, determined empirically. The above procedure would result in the selection of the most important variables.

$$\text{Fittness} = 1/(1 + e^{-(\Sigma v_i w_i)})$$

From amongst the given fitness function $1/(1 + e^{-f})$ is chosen as it results in a value between 0 and 1, which can be perceived as the probability of the selection of variable. Here, v_i is 0 or 1, depending upon if the ith variable is selected or not and w_i is the weight. This is followed by the application of the GA process. The cross over and mutation applied till the population reaches its termination condition. The following algorithm and Figure 2 depicts the process.

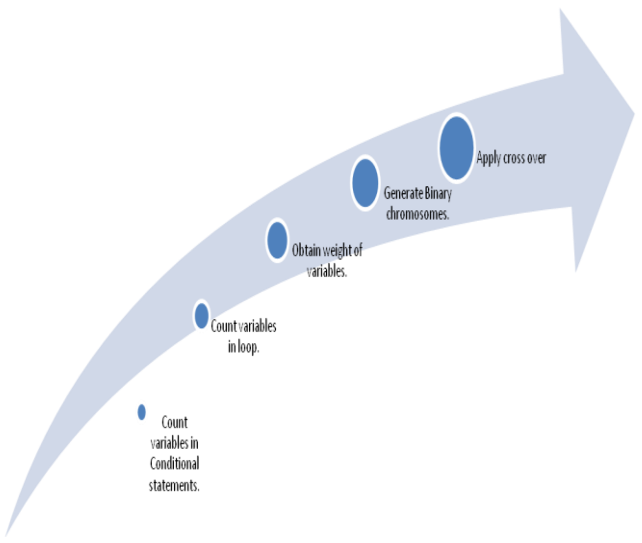
Count variables in Conditional statements.
Count variables in loop.
Obtain weight of variables.
Generate Binary chromosomes.
Apply cross over

Fig- 2: Working model of GABVIE

Our key idea is to take a program code as a input and find all the frequency of all the variables used in the program code. In fig2 two the first steps have been done by using the parser that count the frequency of each and every variable used in the program code. The frequency and the variables have been passed as a parameter in the GABVIE algorithm that evaluates the most important variables. According to that optimal variables, optimal path have been generated.

Algorithm 1: Parser for evaluating the frequency of each variable.
Input: Program statements with variables as parameters
Output: Frequency of each variable

```
For i – 1 to n do
Scan
if equals (t, v[])
then token <- var[ i++]
end if
Scan
If equals (token, block(if))
Scan block(if)
For i – 1 to n
                if equals (token, var [])
                count [] <- count[] +1
                end if
        end for
end if
if equals ( token, block(loop))
        Scan block(loop)
        For i – 1 to m
                if equals (token, var[])
                count <- count +m
                end if
        end for
end if
cell[]  <- var[]
```

Algorithm 2:

Input: Frequency of variables and the vairiables

Output: Frequency of each variable

```
create PopulationChromosome ( count, cell[])
for each cell
if equals (cell [] , 1)
calculate fitnessfunction
fitness <-fit (count [])
return fitness
end if
end for
Select MostfittedChromosome
Point <-RowlettWheel (fitness, PopulationChromosome)
Calculate OnepointCrossover (point)
Calculate Mutation (random (PopulationChromosome))
Repeat until gen <=maxgen
```

The proposed algorithm has been implemented in Python. The IDE used for the implementation is Jupyter. The algorithms scan a C program and perform the tasks as suggested in the algorithm. It may be stated that the program has been divided into two parts. The first part scans the program and finds the frequency of the variables and the second part applies the Genetic Algorithm to find the optimal solution.

VI. FUNCTION USED

Table 1. Presents the most important functions used in the implementation of the algorithm.

Table -1: Important functions used in the implementation (source: self)

Function Name	Arguments	Description
_file	File Name	Opens code file and breaks it into a stream of tokens, returns the stream of tokens in a list named ‘str’.
_var	str	Scans the input stream of code and selects the variables of data type int, char and float except for ‘i’ and ‘j’ which are reserved for loop iterations. Returns a list named ‘var’ containing the names of variables used.
_scan	str	Scans the code and detects the variables that are scanned by the user and takes the input by user. Returns a list ‘scan_var’ with the names of scanned variable and list ‘value’ with the respective value of those scanned variables.
_cou	temp, scan_var, value	‘temp’ contains the for loop condition; This function calculates the number of times a loop iterates. Returns the count of iterations evaluated by the expression in ‘temp’ as variable ‘n’.
_count	str, var, scan_var, value,	Scans the code and calculates the number times a variable is used in the program. Returns a list ‘count’ containing the count of variables in ‘var’

VII. FLOW OF PROGRAM

As stated earlier, the implementation has been divided into two parts, the flow of the two parts have been shown in Figures 3.

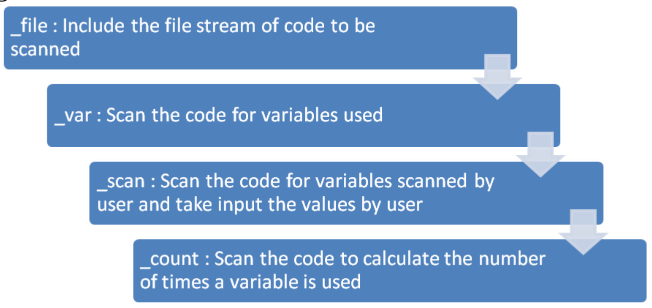
_file : Include the file stream of code to be scanned
_var : Scan the code for variables used
_scan : Scan the code for variables scanned by user and take input the values by user
_count : Scan the code to calculate the number of times a variable is used

Fig-3: Flow of the first part of the implementation

VIII. EXPERIMENTS

The algorithm defined in section 2 has been implemented on the python. The program of calculating the mean, variance and standard deviation is taken for the experiment.

Experiments:

```
void main()
{
int num1, num2, num3, num4, num5;
float mean, var, std;
float dat1, dat2, dat3, dat4 , dat 5;
```

```
printf("\nEnter five numbers\t:");
scanf("%d %d %d %d %d",&num1, &num2, &num3,
&num4, &num5;
mean=(float)(num1+ num2+ num3 + num4 + num5)/5;
var=( (num1-mean)*(num1-mean)
+(num2-mean)*(num2-mean) + (num3-mean)*(num3-mean)
+ (num4-mean)*(num4-mean) + (num5-mean)*(num5
–mean))/5;
std=sqrt(var);
dat1=(num1-mean)/std;
dat2=(num2-mean)/std;
dat3=(num3-mean)/std;
dat4=(num4-mean)/std;
dat5=(num5-mean)/std;
printf("%f %f %f %f %f ",dat1, dat2, dat3, dat4, dat5);
}
```

Table-2: Total number of variables used with their frequency and Fitness (Source: Self)

Number	variable	Frequency	Fitness
1	mean	16	0.999999887
2	std	6	0.997527377
3	num1	4	0.98201379
4	num2	4	0.98201379
5	num3	4	0.98201379
6	num4	4	0.98201379
7	num5	4	0.98201379
8	std	2	0.880797078
9	dat1	2	0.880797078
10	dat2	2	0.880797078
11	dat3	2	0.880797078
12	dat4	2	0.880797078
13	dat5	2	0.880797078

In the Genetic Part, the probability of the variables is to be done on the basis of their fitness .selection of the variables has been done on the basis of having highest fitness would be maximum. Therefore the variable 'mean' is to be selected and has maximum fitness. So if the mean is calculated incorrectly, the standard deviation, variance and scaled values, will all also be incorrect. Same if the std is calculated incorrectly, the variance and scaled values, will all be incorrect.

Genetic Population (The number of cells in each chromosome would be 12 and let us generate 5 such chromosomes).

Table-3: Subsets of chromosomes (Source: self)

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

The corresponding fitness of each chromosome is as follows.

Table -4: Fitness of the variables (Source: self)

1	1.880797
2	2.878324
3	1.761594
4	1.880797
5	1.761594

The Roulette Wheel Selection is then applied. Only those subset will be taken for the next generation whose fitness is high.

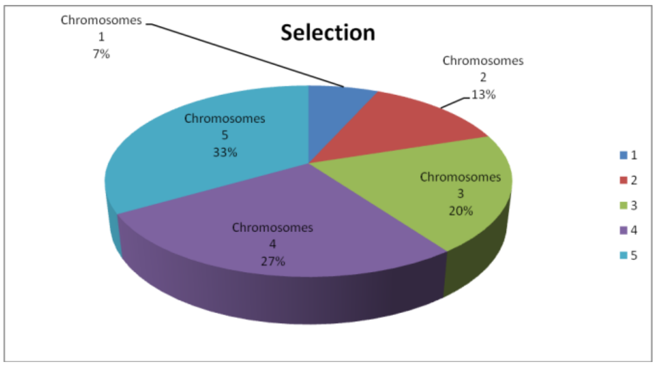

Fig-4:Selection Process

Cross over Operation:

After cross over operation the subset of chromosomes is as follows

Table -5: subsets of chromosomes (Source: self)

Chromosomes	values												
1	1	0	0	0	0	0	0	0	1	0	0	0	0
2	0	1	0	0	0	0	0	1	1	0	0	0	0
3	0	1	0	0	0	0	0	1	0	0	0	1	0
4	1	0	0	0	0	0	0	0	0	0	1	0	0
5	0	0	0	0	0	0	0	0	0	0	1	1	0

Table-6: The fitness of chromosomes after cross over

Chromosomes	fitness
1	1.880796965000
2	3.759121420000
3	3.000000000000
4	1.880796965000
5	1.761594156000

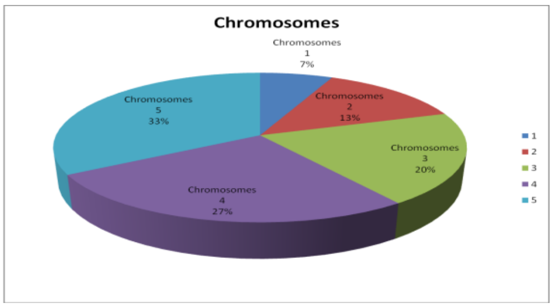

Fig-5: Selection Process after crossover operation

IX. OBSERVATION & RESULTS

The proposed algorithm has been implemented and tested on 7 programs.

The programs included the programs to find the coefficient of correlation, to find Fisher Discriminate Ratio, to find the variance, to find the moment, to find the new centres in K means, to find second maximum and to rotate the line joining given point and origin.

The programs had at maximum 25 variables. In order to select the most revenant variables by brute force, we needed 225 combinations of chromosomes, which amount to 33554432. Each chromosome should have the number of cells equal to the number of variables. This would have been a computationally expensive task.

The most important variables have been found by the proposed algorithm. The chromosomes were initially generated randomly. However, it was aimed at finding out the top 20% variables. For a given crossover rate and mutation rate, it was found that the proposed technique was able to give 90.714% of correct results.

Table -7: Frequency calculated by parser and manually (Source: self)

Number of variables	The most important variables found by technique	The most important variables found manually	Percentage
25	4	5	80
25	5	5	100
24	5	5	100
25	4	5	80
20	3	4	75
20	4	4	100
20	3	4	100

This was followed by the variation of cross over rate from 5 to 10 percent. For each cross over rate the best result was noted. The experiment resulted in the increase in the percentage of correct variables selected to 93.57143, as shown in the following table.

Table -8: Comparison of frequency of variables manually with the parser (Source: self)

Number of variables	The most important variables found by technique	The most important variables found manually	Percentage
25	5	5	100
25	5	5	100
24	5	5	100
25	4	5	80
20	3	4	75
20	4	4	100
20	3	4	100

This was followed by the variation of mutation rate from 2 to 5 percent. For each mutation rate the best result was noted. The experiment resulted in the increase in the percentage of correct variables selected to 97.14286, as shown in the following table.

Table -9: Comparison of frequency of variables manually with the parser after crossover operation (Source: self)

Number of variables	The most important variables found by technique	The most important variables found manually	Percentage
25	5	5	100
25	5	5	100
24	5	5	100
25	4	5	80
20	4	4	100
20	4	4	100
20	3	4	100

The complete analysis results in the selection of most of the important variables found manually. It may be stated that the number of experiments carried out for cross over were 11 as the cross over rates 5, 5.5, 6, 6,5 etc. were taken. Likewise, the number of experiments carried out for varying mutation rate was 7. The mutation rate was varied from 2 to 5 in the gap of 0.5. Theses 18 experiments were carried out for 7 programs hence 126 experiments were carried out to find the optimal solution.

- It was noted that variation of mutation rate has more effect on the technique.
- At mutation arte =3.5 best answers were obtained.
- If the cross over rate was 8.5 best answers were obtained.

X. FUTURE SCOPE

The following empirical analysis is being carried out to find better solution.

- Variation of the types of cross over
- Variation of the types of mutation
- Variation of the selection technique

Moreover the technique would be applied to larger programs.

REFERENCES

1. Abreu, B. T. D., Martins, E., Sousa, F. L. D. 2007. Generalized extremal optimization: an attractive alternative for test data generation. GECCO 2007. 1138.
2. Angeline, P. J., Michalewicz, Z., Schoenauer, M., Yao, X., Zalzala A. 1999. The Pareto Archived Evolution Strategy: A New Baseline Algorithm for Pareto Multiobjective Optimisation, IEEE Press: Mayflower Hotel, Washington D.C., USA. Vol.1.
3. Bhasin, H., Singla, N. 2012. Harnessing Cellular Automata and Genetic Algorithms To Solve Travelling Salesman Problem. International Conference on Information, Computing and Telecommunications, (ICICT -2012). 72 – 77.
4. Blanco, R., Tuya, J., Adenso-Díaz, B. 2009. Automated test data generation using a scatter search approach. Information and Software Technology, 51, 4, 708-720.
5. Clarke, L. A. 1976. A system to generate test data and symbolically execute programs. IEEE Trans. Sofrware Eng. SE-2, 3, 215-222.
6. Deb, K., Pratap, A., Agarwal, S., Meyarivan, T. 2002. A fast and elitist multiobjective genetic algorithm : NSGA-II. IEEE Transactions on Evolutionary Computation. 6, 2, 182–197. [7] Edvardsson, J. 1999. A Survey on Automatic Test Data Generation. In Proceedings of the Second Conference on Computer Science and Engineering in Linkoping. 21-28.
7. Engström, E., Runeson, P., Skoglund, M. 2010. A systematic review on regression test selection techniques. Information and Software Technology.
8. Ferrer, J., Kruse, P., Chicano, F., Alba, E. 2012. Evolutionary algorithm for prioritized pairwise test data generation. GECCO 2012. 1213-1220.
9. Ferrer, J., Chicano, F. and Alba, E. 2012. Evolutionary algorithms for the multi-objective test data generation problem. Softw: Pract. Exper. 42, 1331–1362.
10. Floreano, D., Mattiussu, C. 2008. Bio – Inspired Artificial Intelligence: Theories, Methods, and Technologies. MIT Press.
11. Girgis, M.R. 2005. Automatic test data generation for data flow testing using a genetic algorithm. Journal of Universal Computer Science. 11, 5, 898–915.

12. Gong, D., Zhang, W., Yao, Y. 2011. Evolutionary generation of test data for many paths coverage based on grouping. Journal of Systems and Software. 84, 12, 2222-2233.
13. Harman, M., Lakhotia, K., McMinn, P. 2007. A multi-objective approach to search-based test data generation. GECCO ’07: Proceedings of the 9th annual conference on Genetic and evolutionary computation, ACM: New York, NY, USA. 1098– 1105.
14. Harman, M., Kim, S. G., Lakhotia, K., McMinn, P., Yoo, S. 2010. Optimizing for the number of tests generated in search based test data generation with an application to the oracle cost problem. Proceedings of the 3rd International Workshop on Search-Based Software Testing (SBST) in conjunction with ICST 2010, IEEE: Paris, France. 182–191.
15. Jones, B., Sthamer, H., Eyres, D. 1996. Automatic structural testing using genetic algorithms. Software Engineering Journal. 11, 5, 299–306.
16. Lin, J., Yeh, P. 2001. Automatic test data generation for path testing using GAs. Information Sciences: an International Journal. 131, 1-4, 47-64.
17. Malhotra, R., Garg, M. 2011. An Adequacy Based Test Data Generation Technique Using Genetic Algorithms. Journal of Information Processing System (JIPS), 7, 2, 363-384.
18. McMinn, P. 2004. Search-based software test data generation: a survey. Research Articles, Software Testing, Verification & Reliability. 14, 2, 105-156.
19. Michael, C.C., McGraw, G.E., Schatz, M.A. 2001. Generating software test data by evolution. IEEE Transactions on Software Engineering. 27, 12, 1085–1110.
20. Miller, J., Reformat, M., Zhang, H. 2006. Automatic test data generation using genetic algorithm and program dependence graphs. Information and Software Technology. 48, 7, 586–605.
21. Miller, W. and Spooner, D. L. 1976. Automatic generation of floating-point test data. IEEE Trans. Software Eng. SE-2, 3, 223226.
22. Nebro, A. J., Durillo, J. J., Luna, F., Dorronsoro, B., Alba, E. 2009. MOCell: A cellular genetic algorithm for multiobjective optimization. Int. J. Intell. Syst. 24, 7, 726–746.
23. Neumann, J. V. 1996. Theory of Self-Reproducing Automata. University of Illinois Press, Champaign, IL.
24. Pargas, R.P., Harrold, M.J., Peck, R.R. 1999. Test data generation using genetic algorithms. Journal of Software Testing, Verification and Reliability. 9, 4, 263–282.
25. Pesavento, U. 1995. An implementation of von Neumann's selfreproducing machine. Artificial Life, 2, 4, 337-354.
26. Sofokleous, A. A., Andreou, A. S. 2008. Automatic, evolutionary test data generation for dynamic software testing. Journal of Systems and Software. 81, 11, 1883-1898.
27. Sthamer, H. 1996. The automatic generation of software test data using genetic algorithms. Ph.D. Thesis, University of Glamorgan, Pontypridd, Wales, UK.
28. Ramamoorthy, C. V., Ho, S. and Chen, W. T. 1976. On the automated generation of program test data. IEEE Trans. Software Eng. SE-2, 4, 293-300.
29. Watkins, A. 1995. The automatic generation of test data using genetic algorithms. Proceedings of the 4th Software Quality Conference, 300–309.
30. Watkins, A., Hufnagel, E.M. 2006. Evolutionary test data generation: a comparison of fitness functions. Software Practice and Experience. 36, 95–116.
31. Weisstein, Eric W. “Cellular Automaton.” From MathWorld--A Wolfram Web Resource. http://mathworld.wolfram.com/CellularAutomaton.html.
32. Wegener, J., Baresel, A., Sthamer, H. 2001. Evolutionary test environment for automatic structural testing. Journal of Information and Software Technology. 43, 14, 841–854.
33. Wolfram, S. 1994. Cellular Automata and Complexity: Collected Papers, ISBN 0-201-62716-7.
34. Xanthakis, S., Ellis, C., Skourlas, C., Le Gall, A., Katsikas, S. 1992. Application of genetic algorithms to software testing. 5th International Conference on Software Engineering and its Applications, Toulouse, France. 625–636.
35. Yoo, S., Harman, M. 2012. Regression testing minimization, selection and prioritization: a survey. Softw. Test., Verif. Reliab. 22, 2, 67-120.
36. Zitzler, E., Laumanns, M., Thiele, L., 2001. SPEA2: Improving the strength pareto evolutionary algorithm. Technical Report 103, Gloriastrasse 35, CH-8092 Zurich, Switzerland.
37. P. McMinn, M.Holcombe(2003), “The State Problem for Evolutionary Testing”, GECCO, pp. 2488–2498

AUTHORS PROFILE

Ms Seema Sharma, is currently working as Assistant Professor in Faculty of Computer Applications, Manav Rachna International Institute of Research and Studies, Faridabad. She has 12+ years’ experience of teaching undergraduate and post graduate students. She has published more than 8 research papers in International conferences and participated in many National/ International Conferences to her credit. Her research interests are in the areas of Software Engg, Machine Learning and Software Testing. She has also written a book on Data Structures.

Dr. Shaveta Bhatia has been awarded her Ph.D degree in Computer Applications. She has completed her Master in Computer Applications (MCA) from Kurukshetra University. She is having 17 years of academic and research experience. She is a member of ACM, IAENG and CSI. She has participated in various National and International Conferences and actively involved in various projects. There are more than 30 publications to her credit in reputed National and International Journals and Conferences. She is also member of Editorial board of various highly index journals. Her specialized domains include Web Applications and Software Engineering and guiding research scholars in these areas