Effective Software Fault Localization using an Back Propagation Neural Network

Abha Maru, Arpita Dutta, Vinod Kumar and Durga Prasad Mohapatra

Abstract Effective fault localization is an essential requirement of software development process. Back-propagation neural network can be used for localizing the faults effectively and efficiently. Existing NN based fault localization techniques take statement invocation information in *binary terms* to train the network. In this paper, we have proposed an efficient approach for fault localization using back propagation neural network and we have used the actual number of times the statement is executed to train the network. We have investigated our approach on Siemens suite. Results show that on an average there is 35% increase in the effectiveness over existing BPNN.

Key words: Back-Propagation neural network; program debugging; Fault localization; Suspiciousness of code; failed test; successful test.

1 Introduction

Software has become a key part of our society. Various softwares are being developed every year and requires huge amount of efforts. This effort lies not only in development but majorly in testing. Debugging process becomes more complex when scale and complexity of software increases. The stage in which bugs are identified is known as fault localization. To make the software fault free we must first need to locate the bug. This process can be done in two stages. In first stage, identifying the code which might contain the bug, called as suspicious code. In second stage, the programmer reviews the selected code and tries to identify the bug [30].

Currently, many research work has been done in the field of fault localization such as, Baah et al. [28] [8] [24]proposed probabilistic program dependence graph (PPDG), Slice-based techniques such as static slicing, dynamic slicing and execution slicing [3–6,10–13,16,18,19], spectrun based techniques [14] such as Tarantula [17], Ochiai [22], Visualization [15], SOBER [27], Crosstab (CBT) [29] etc. DStar(D*), proposed by Wong et al. [26], is currently the state of the art for fault localization. Its effectiveness was evaluated using 24 different programs and it performs better that other 38 techniques for fault localization.

Nowadays, machine learning techniques such as artificial neural networks [21], deep neural networks [25] are widely used to get a robust model for fault localization. Advantages of neural networks over other models have enticed many researchers. Fault tolerant characteristic of neural network makes it adaptive to the working environment and minor faults have very little impact on the model. This has made it applicable to many software engineering areas like reliability estimation, risk analysis, cost estimation and reusability characterization etc.

Back propagation neural network (BPNN) model for fault location, proposed by Wong et al in [20]. is one of the most popular model in today's scenario. Local minima was the disadvantage of the BP neural network. Thus,

Abha Maru
National Institute of Technology, Rourkela, Odisha, Pin: 769008
e-mail: 217cs3313@nitrkl.ac.in

Arpita Dutta
Indian Institute of Technology, Kharagpur, West Bengal, Pin: 721302
e-mail: arpitad10j@gmail.com

Vinod Kumar
National Institute of Technology, Rourkela, Odisha, Pin: 769008
e-mail: 517cs1007@nitrkl.ac.in

Durga Prasad Mohapatra
National Institute of Technology, Rourkela, Odisha, Pin: 769008
e-mail: durga@nitrkl.ac.in

another method for fault localization was proposed by Wong et al. in [23], using radial basis function (RBF) network. RBF and other spectrum based techniques are handicapped with the problem of statements having same suspiciousness, whereas, BPNN model assigns unique suspiciousness score to each executable statement.

In the existing back propagation neural network (BPNN) model, execution of statement is recorded in binary terms. This over looked those statements which are executed more than once which mislead the results as there is a probability that the statement which is executed more than once is more critical than the statement which is executed only once. This motivated us to work upon the existing BP neural network model to improvise the results. Thus, to improve the feature vector for training the network, the proposed method uses the actual number of times the statement is executed for the given test case. We have experimented with five different programs of Siemens suite to analyze the effectiveness of the proposed approach.

The remainder of the paper is organized as follows. Section 2 explains the basic concepts of the artificial neural network and back propagation neural network. Section 3 explains the proposed approach of fault location using BP neural network along with example. Section 4 explains the experimental setup, data set used, results and average results. Section 5 shows the comparison with related work. Section 6 explains the threats to validity of our approach. Section 7 gives the conclusion and tells the future work which can be done.

2 Basic Concepts

In this section we explain the basics of artificial neural network, their advantages over other conventional methods and brief overview of back propagation neural network.

2.1 Artificial Neural Network

Artificial neural network (ANN) is viewed as a mathematical model inspired by the functional aspects of biological nervous system. The network consists of the simple elements called as neurons and their working is similar to that of biological neurons. Each neuron consist of certain function called as activation function which converts the input to the desired output form. Network consists of multiple layers and each layer contains multiple neurons which work in parallel to generate the output. Its learning capability can approximate any non-linear continuous functions based on the given data. Neurons are connected with each other with certain weights upon the connections, so that they can process the information collectively and store it on these weights. These networks are very useful as they can learn from past experience and using this knowledge can solve new related problems.

2.2 Back Propagation Neural Network

A back propagation neural network is a kind of feed forward neural network in which the neurons are divided into layers. Neurons of one layer can be only connected to the neurons of the next layer. Layers between the input and output layer are called as hidden layers. Back propagation algorithm is an iterative process used to adjust the errors while training the network.The three layered structure of BP neural network is shown in Figure 1.

3 Proposed Approach

In this section, we discuss our proposed approach for fault localization using back propagation neural network.

3.1 Fault Localization using BP neural network

Initially, a program P is taken as an base input which has m number of executable statements along with one faulty statement. Suppose P is executed on n test cases, out of which k test cases are passed and n-k are failed. Figure 2 shows the sample *coverage matrix* in which number of rows is equal to the number of test cases and number of columns is equal to the number of statements covered by the test cases. Value of each row is

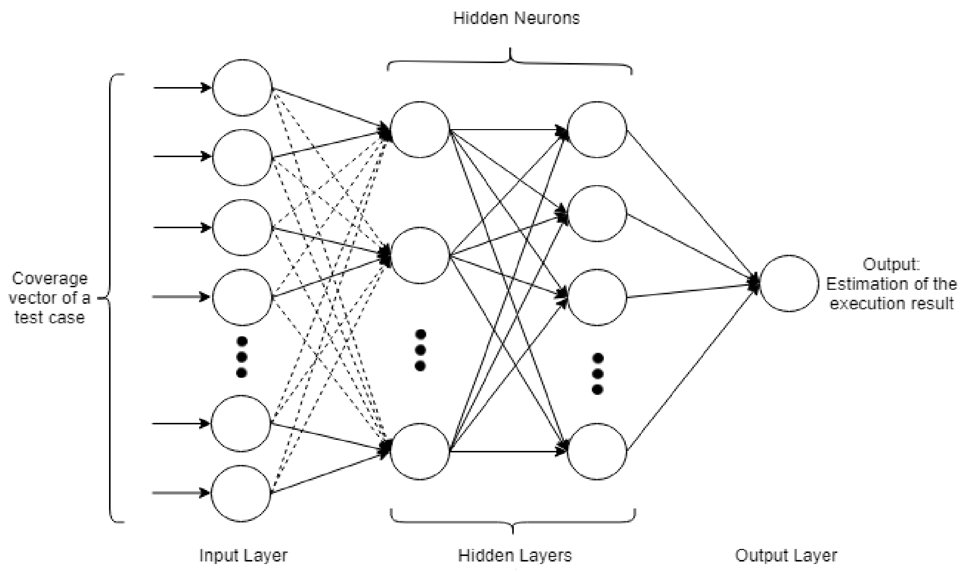
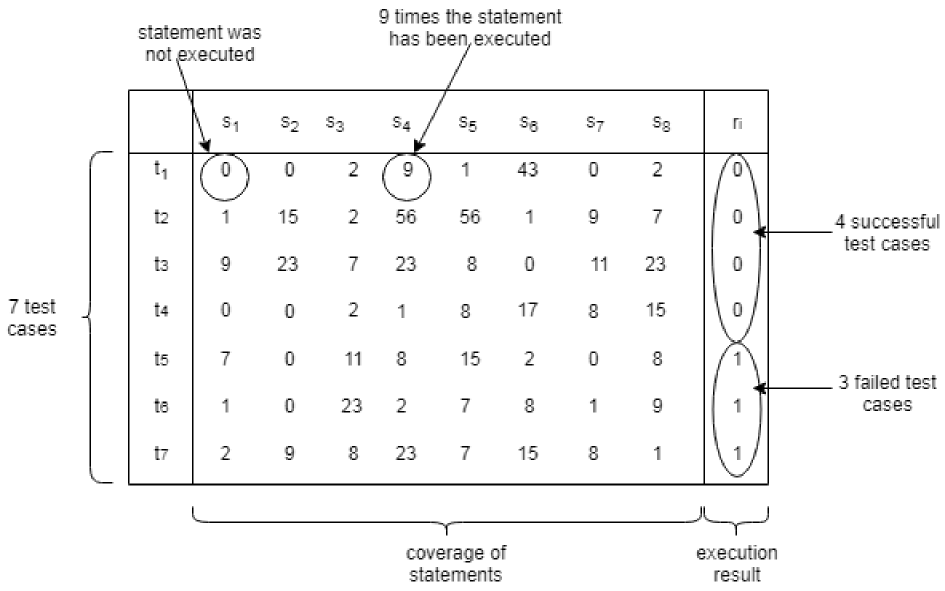

Hidden Neurons
Coverage vector of a test case
Output: Estimation of the execution result
Input Layer
Hidden Layers
Output Layer

Fig. 1 Structure of a three layered BP Neural Network.

the statement coverage of a particular statement corresponding to the particular test case. Any real number greater than zero indicates the number of times the statement is executed for that particular test case and zero indicates that the statement was not executed by that corresponding test case. For example, s_4 is covered by the successful test case t_1 and s_2 is not covered by the failed test case t_6. Complete row can be called as a *coverage vector (c_{t_i})* which depicts the coverage information for test case t_i. For example, c_{t_1} = (0,0,2,9,1,43,0,2) is the first row of the matrix in the Figure 2 shows the statement coverage of the program for test case t_1.

Now, assuming a set of *virtual test cases* v_1, v_2, v_3,...,v_m. Depicted in a form of matrix called as *virtual matrix*


statement was not executed
9 times the statement has been executed
4 successful test cases
3 failed test cases
7 test cases
coverage of statements
execution result

	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8	r_i
t_1	0	0	2	9	1	43	0	2	0
t_2	1	15	2	56	56	1	9	7	0
t_3	9	23	7	23	8	0	11	23	0
t_4	0	0	2	1	8	17	8	15	0
t_5	7	0	11	8	15	2	0	8	1
t_6	1	0	23	2	7	8	1	9	1
t_7	2	9	8	23	7	15	8	1	1

Fig. 2 Sample coverage matrix.

as shown in the Equation 1, whose each row is the *virtual coverage vector* c_{v_1},...,c_{v_m}. Specialty of the virtual test case is that, it executes only one statement, which is not possible in reality, hence they are called as virtual test cases. Each row of this matrix is given as an input to the proposed BP neural network and output generated is the suspiciousness of corresponding statement.

$$\begin{bmatrix} c_{v_1} \\ c_{v_2} \\ c_{v_3} \\ \vdots \\ c_{v_m} \end{bmatrix} = \begin{bmatrix} 1\,0\,0 \cdots 0 \\ 0\,1\,0 \cdots 0 \\ 0\,0\,1 \cdots 0 \\ \vdots\,\vdots\,\vdots\, \ddots\, \vdots \\ 0\,0\,0 \cdots 1 \end{bmatrix} \tag{1}$$

3.2 Explanation with Example

Let us now take an example to make our proposed model more clear. Suppose we have a program with 20 statements having 8 executable statements (m=8) with one fault at line no 7. We took a test suit of 7 test cases, out of which three test cases failed. Statement coverage and result of execution is shown in Figure 2. After this following steps are performed.

- Firstly, a BP neural network is created having 8 input neurons, one output neuron, and three hidden layers of neurons. We considered sigmoid function as *non-linear transfer function.*
- Then, network is trained with the coverage data collected. First input in our example would be vector (0,0,2,9,1,43,0,2) with zero as the expected output. If the actual output of the network does not match with the expected output, then the weight of the network is updated. Then, this updated weight is used for next input to train the network. This whole process is repeated several times till the error of the network becomes minimal(in our case approx 0.001).
- After the training is done, BP neural network is fed with the test data. In our case, coverage vector of the virtual test cases is the test input data. Suspiciousness is the likelihood of the statement containing the fault. Output generated gives the suspiciousness of each statement, as shown in the Table 1.
- Arranging the suspiciousness values in decreasing order, we found the following order s_5, s_2, s_1, s_7, s_6, s_8, s_4, s_3. When we examined each statement one by one, s_7 which is the actual faulty statement, is evaluated after examining 3 non faulty statements. Here, in our example, rank of s_7 is 4.

Table 1 Actual output(suspiciousness)generated by BPNN for each statement.

Statement	Output	Statement	Output
s_1	0.65311	s_5	0.81204
s_2	0.75211	s_6	0.60274
s_3	0.65311	s_7	0.62310
s_4	0.29561	s_8	0.46922

4 Experiments

In this section, we present the experimental setup and results of our experiments.

4.1 Experimental setup

We have implemented our approach, in Python-3, on Ubuntu 16.04 operating system, with i5 64bit processor having 4GB RAM. For experiment, values of learning rate is taken in the range of 0.0001 to 1 and number of hidden layers is calculated heuristically using Equation 2.

$$num = round(n/30) * 0.5 \tag{2}$$

Here, n represents the number of executable statements in the program. We have used, Sigmoid function as the activation function (transition function)of the BP neural network.

4.2 Data set used

For experimental use five C programs and their corresponding faulty versions of Siemens suit were used, namely Print_tokens, Print_tokens2, Schedule, Schedule2, Tot_info. Each faulty version contains only one fault. Table 2 introduces the 5 C programs with their names, number of faulty versions, lines of code, number of executable statements and number of test cases. Siemens suit contains in total 7 programs out of which we have selected

Table 2 Details of Siemens suit's programs.

Name of program	Number of faulty versions	Lines of Codes	Number of executable statements	number of test cases
Print_tokens	7	565	172	4130
Print_tokens2	10	510	146	4115
Schedule	9	412	140	2650
Schedule2	10	307	115	2710
Tot_info	23	406	100	1052

only 46 faulty versions of 5 C programs. Following versions were omitted: version 4 and 6 of Print_tokens, version 10 of Print_tokens2, version 6, 10 and 21 of Tot_info, version 1, 5, 6, 8 and 9 of Schedule and version 8 and 9 of Schedule2. Reasons of omission were:

- Syntactic difference was not found between the correct version and faulty versions (e.g., included header files were different).
- None of the test cases failed for the faulty versions.
- Segmentation faults were observed while executing the test cases on the above omitted faulty versions.
- Difference between the actual program and faulty versions was not included in the executable statements. Additionally, certain statements were absent in the faulty versions which cannot be located directly.

4.3 Results

In this section we present our experimental results using different graphs. Figure 3, shows the effectiveness of proposed approach for Print_tokens, such that 72% of the program needs to be evaluated for locating the fault as compared to existing BPNN model [20] in which 85% of the program is evaluated. Figure 4, shows that for Print_tokens2, on an average 18% of the program needs to be evaluated for locating the fault as compared to existing BPNN model in which 36% of the program is evaluated, although in the graph we can see that near 10% to 16% our approach does not show good results but on an average result is good. Figure 5, shows that for Schedule, 20% of the program needs to be evaluated for locating the fault as compared to existing BPNN model in which 50% of the program is evaluated. Figure 6, shows that for Schedule2, 38% of the program needs to be evaluated for locating the fault as compared to existing BPNN model in which 84% program is evaluated. Figure 7, shows that for Tot_info, 52% of the program needs to be evaluated for locating the fault as compared to existing BPNN model in which 80% of the program is evaluated.

We have also used EXAM score metric to measure the effectiveness of various techniques. EXAM score is calculated using Equation 3.

$$EXAM\ Score = \frac{|V_{examined}| * 100\%}{|V|} \tag{3}$$

Where,

- |V| measures the size of executable codes in the program.
- $|V_{examined}|$ measures the number of statements that has to be inspected so as to find the fault.

Figure 8, shows the comparison of average EXAM score of the existing BPNN model [20] and the proposed BPNN model. Table 3, shows the average percentage of increase in the effectiveness of proposed BPNN model for fault localization as compared to the existing BPNN model. Overall, we can say that our model has improved 35% of the effectiveness on an average.

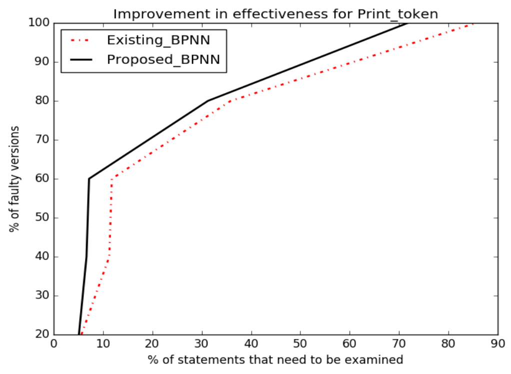


Fig. 3 Improvement in effectiveness for Print_tokens.

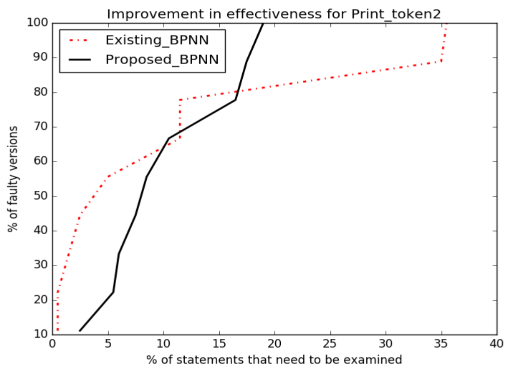


Fig. 4 Improvement in effectiveness for Print_tokens2.

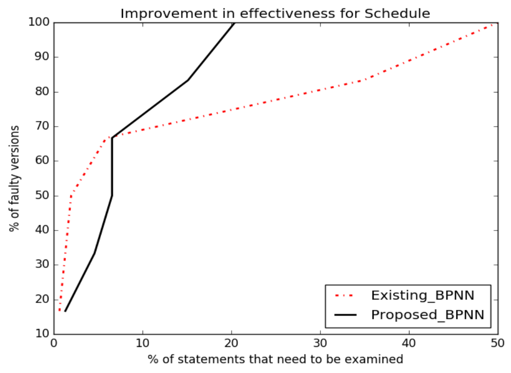


Fig. 5 Improvement in effectiveness for Schedule.

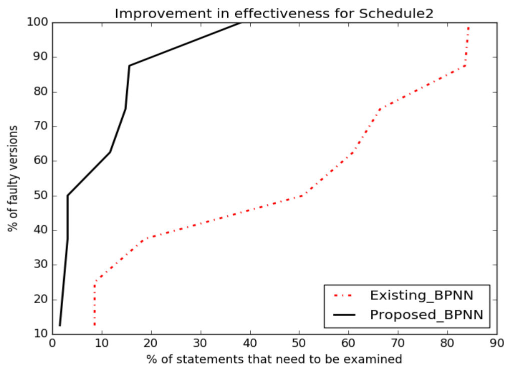


Fig. 6 Improvement in effectiveness for Schedule2.

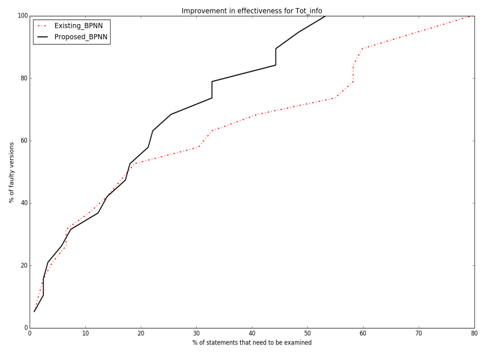


Fig. 7 Improvement in effectiveness for Tot_info.

Table 3 Average percentage of effectiveness for each program.

Program	Avg EXAM score (Existing)	Avg EXAM score (Proposed)	Percentage of increase in effectiveness (%)
Print_tokens	30.05	24.41	18.77
Print_tokens2	11.5	10.38	9.66
Schedule	15.78	9.10	42.36
Schedule2	47.75	11.32	76.27
Tot_info	29.81	21.48	27.93

5 Comparision with related work

Many work have been done previously in the field of fault localization.
Baah et al. [28] proposed probabilistic program dependence graph(PPDG), which provides the conditional probability of each node in the graph to identify the bug [8,24]. But this becomes a lengthy process as complete program is considered for debugging. Slice-based techniques [3–6, 10–13, 16, 18, 19] were also adopted, which

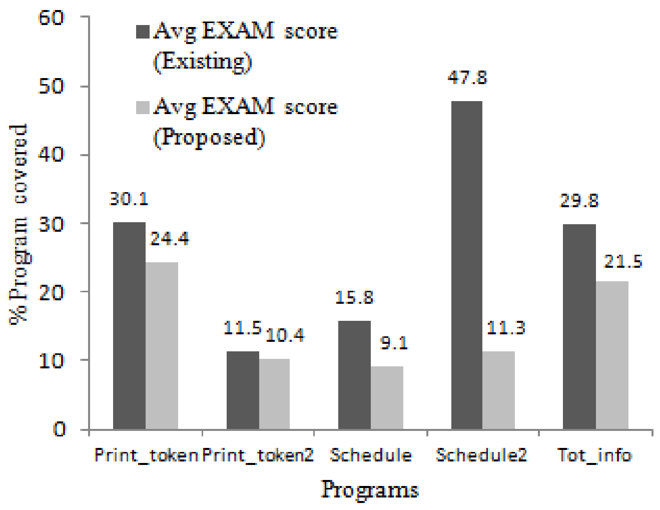


Fig. 8 Comparison of average EXAM score of existing BPNN and proposed BPNN model.

narrows down the search to various slices of the program. Slice is basically a set of statements in a program which leads to the faulty statement or the set of statements which can be impacted by the faulty statement [1] [2].

Disadvantage of earlier slice based techniques is that, they only inspect the slice, leaving the remaining part of the code. Moreover, sometimes the slices include all the statements of the program, which again leads to the initial problem state. In our approach, we rank all the statements and based upon the unique ranking of the statements, they are evaluated.

Collofello suggested spectrum based techniques which are based on mathematical formulas. Jones et al. [17] proposed a popular (ESHS)-based technique, called Tarantula. It utilized the pass/fail information about each test case. The main idea behind it was, the entities which are executed by the failed test cases have more probability of having fault than those which are executed by the passed test cases. Jones et al. [15]extended Tarantula by representing degree of suspiciousness by using different colors . Liu et al. [27] proposed a method called SOBER, which ranked suspicious predicates by calculating the difference between the truth value of predicates for successful and failed execution. Wong et al. [29] proposed an analysis based technique called Crosstab(CBT) which provides the rank to the suspicious elements of the program.

Spectrum based fault localization techniques failed when there were statements having same degree of suspiciousness. Another weakness that they only examined the suspicious statements. However, there is a possibility of bug containment in the statements covered by the successful test cases. Whereas, in our approach all the statements are given ranks based upon their unique suspiciousness, so there is no conflict.

A Back propagation neural network (BPNN) model for fault location, was proposed by Wong et al. [20]. In this model, for each test case, statement coverage along with the result obtained after execution were collected. This collected data was used as input to train the network. This provides the network, the ability to relate both the entities. The output generated by the network is the suspiciousness of each statement for being faulty. Since, local minima was the disadvantage of BP neural network, another method for fault localization was proposed by Wong et al. [23], using radial basis function (RBF) network, which is less sensitive to those problems. RBF and other spectrum based techniques are handicapped with the problem of statements having same suspiciousness, whereas, BPNN model assigns unique suspiciousness score to each executable statement.

In the existing back propagation neural network (BPNN) model, execution of statement is recorded in binary terms. This over looks those statements which are executed more than once. In our technique, we used the number of times the statement is executed as an input to the BP neural network, which helped in getting the exact value of the suspiciousness for each statement.

6 Threats to validity

Though our proposed method is performing better than the existing BPNN model but it has certain constraints.

- Our model has been tested for procedural C program, so object oriented aspects have not been tested till yet.
- Programs used in our approach had only single faults. Thus, effectiveness our our model over programs having multiple faults have not been tested.

- Since we are considering actual number of statement invocation in the coverage matrix, we require those code which has loops, recursive calls, function calls etc. Then only statement will be executed more than once. Programs which do not have these features cannot be used for checking the effectiveness of our approach.
- We have used standard sample programs for our experiments whose length is less than thousand lines of code. We cannot assure the effectiveness of our approach over very large sized programs, as we have not experimented on them.

7 Conclusion and Future Work

We have extended the existing BP neural network model for fault localization by using the actual number of times statement execution in the coverage matrix. For Print_token 18.77% increase , Print_token2 9.66% increase, for Schedule 42.36% increase, Schedule2 76.27% increase and for tot_info 27.93% increase in the effectiveness has been seen. Though for Print_token and Print_token2 improvement in the effectiveness is not much, but on an average we are able to improve the effectiveness by 35% using our proposed approach for fault localization using BP neural network as compared to the existing BP neural network model.

In future we can extend this work by changing various activation function and can check for the increase of effectiveness. Programs having multiple faults can be tested. Programs having object oriented features can be taken. Programs used in industries which are of very large size and highly complex in nature can also be used to test the effectiveness of our approach. Since we tested on C programs, other languages such as Java, Python, C#, Perl, etc can be used. This approach of using actual number of statement execution can also be applied on other machine learning models like RBF, SVM etc and comparison can be done.

References

1. G. B. Mund, D. Goswami, R. Mall, *Program Slicing, The Compiler Design Handbook.* CRC Press, pp. 269-294, 2002.
2. J. Krinke, *Slicing, chopping, and path conditions with barriers.* Softw. Quality Control, 12(4), pp. 339–360, Dec. 2004.
3. M. Weiser, *Program slicing.* IEEE Transactions on Software Engineering, SE-10(4):352-357, July 1984.
4. J. R. Lyle and M. Weiser, *Automatic Program Bug Location by Program Slicing.* In Proceedings of the 2 nd International Conference on Computer and Applications, pp. 877-883, Beijing, China, June 1987.
5. B. Korel and J. Laski, *Dynamic Program Slicing*, Information Processing Letters, 29(3), pp. 155-163, October 1988.
6. H. Agrawal and J. R. Horgan, *Dynamic Program Slicing.* in Proceedings of the ACM SIGPLAN'90 Conference on Programming Language Design and Implementation, pp. 246- 256, White Plains, New York, June 1990.
7. B. Korel, *PELAS – Program Error-Locating Assistant System.* IEEE Transactions on Software Engineering, 14(9), pp. 1253-1260, September 1988.
8. A. B. Taha, S. M. Thebaut, and S. S. Liu, *An Approach to Software Fault Localization and Revalidation based on Incremental Data Flow Analysis.* in Proceedings of the 13 th Annual International Computer Software and Applications Conference, Washington DC, USA, pp. 527-534, September 1989.
9. H. Agrawal, R.A. DeMillo, and E.H. Spafford, *An Execution Backtracking Approach to Program Debugging.* IEEE Software, 8(5), pp. 21–26, May 1991.
10. H. Agrawal, R. A. DeMillo and E. H. Spafford , *Debugging with Dynamic Slicing and Backtracking.* Software Practice and Experience, 23(6), pp. 589-616, June, 1993.
11. R. Gupta and M. L. Soffa, *Hybrid slicing: an approach for refining static slices using dynamic information.* in Symposium on Foundations of Software Engineering, pp. 29–40, 1995.
12. F. Tip, *A survey of program slicing techniques.* Journal of Programming Languages, 3(3), pp. 121–189, 1995.
13. H. Agrawal, J. R. Horgan, S. London, and W. E. Wong, *Fault Localization using Execution Slices and Dataflow Tests.* in Proceedings of the 6 th IEEE International Symposium on Software Reliability Engineering, pp. 143-151, Toulouse, France, October 1995.
14. M. J. Harrold, G. Rothermel, K. Sayre, R. Wu, and L. Yi, *An Empirical Investigation of the Relationship between Spectra Differences and Regression Faults.* Journal of Software Testing, Verification and Reliability, 10(3), pp. 171-194, September 2000.
15. J. A. Jones, M. J. Harrold, and J. Stasko, *Visualization for fault localization.* in Proc. Workshop Softw. Vis., 23rd Int. Conf. Softw. Eng., Ontario, BC, Canada, pp. 71–75, May 2001.
16. X. Zhang, H. He, N. Gupta, and R. Gupta, *Experimental Evaluation of Using Dynamic Slices for Fault Location.* in Proceedings of the 6 th International Symposium on Automated Analysis-driven Debugging, pp. 33-42, Monterey, California, USA, September 2005.
17. J. A. Jones and M. J. Harrold, *Empirical Evaluation of the Tarantula Automatic Fault- Localization Technique.* in Proceedings of the 20 th IEEE/ACM Conference on Automated Software Engineering, pp. 273-282, Long Beach, California, USA, December, 2005.
18. X. Zhang, S. Tallam, N. Gupta, and R. Gupta, *Towards Locating Execution Omission Errors.* in Proceedings of the 2007 ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI), pp. 415-424, San Diego, California, USA, June 2007.
19. C. Liu, X. Zhang, J. Han, Y. Zhang, and B.K. Bhargava, *Indexing Noncrashing Failures: A Dynamic Program Slicing-Based Approach.* in Proceedings of the 23 rd International Conference on Software Maintenance, pp. 455-464, Paris, France, October 2007.

20. W. E. Wong and Y. Qi, *BP Neural Network-based Effective Fault Localization.* International Journal of Software Engineering and Knowledge Engineering, 19(4), pp. 573-597, June 2009.
21. S. Nessa, M. Abedin, W. Eric Wong, L. Khan, and Y. Qi,*Fault Localization Using N-gram Analysis*, in Proceedings of International Conference on Wireless Algorithms, Systems, and Applications, pp. 548-559, October 2008.
22. L. Naish, L. Hua Jie, and R. Kotagiri, *A model for spectra-based software diagnosis.* ACM Transactions on software engineering and methodology (TOSEM) 20, 11(3), 2011.
23. W. E. Wong, V. Debroy, R. Golden, X. Xu, and B. Thuraisingham, *Effective software fault localization using an RBF neural network*, IEEE Transactions on Reliability,61(1), pp. 149–169, March 2012.
24. F. Deng, J.A. Jones, *Weighted System Dependence Graph.* Software Testing, Verification and Validation (ICST), 2012 IEEE Fifth International Conference on, pp. 380-389, 17-21 April 2012.
25. Z. Zhang, L. Yan, T. Qingping, M. Xiaoguang, Z. Ping, and C. Xi. *Deep Learning-Based Fault Localization with Contextual Information.* IEICE Transactions on Information and Systems, 100(12), pp. 3027-3031, 2017.
26. Wong, W. Eric, Vidroha Debroy, Ruizhi Gao, and Yihao Li. ”The DStar method for effective software fault localization.” IEEE Transactions on Reliability 63, no. 1 (2014): 290-308.
27. C. Liu, X. Yan, L. Fei, J. Han, and S. P. Midkiff. SOBER: statistical model-based bug localization. In ESEC/FSE’05
28. Baah, George K., Andy Podgurski, and Mary Jean Harrold. ”The probabilistic program dependence graph and its application to fault diagnosis.” IEEE Transactions on Software Engineering 36, no. 4 (2010): 528-545.
29. Wong, E., Wei, T., Qi, Y. and Zhao, L., 2008, April. ”A crosstab-based statistical method for effective fault localization”. In Software Testing, Verification, and Validation, 2008 1st International Conference on (pp. 42-51). IEEE.
30. Wong, W. Eric, Ruizhi Gao, Yihao Li, Rui Abreu, and Franz Wotawa. ”A survey on software fault localization.” IEEE Transactions on Software Engineering 42, no. 8 (2016): 707-740.