An Empirical Study On Fault Localization And Effective Test Case Selection By Neural Network

A. Pravin
Research Scholar, Sathyabama University, Chennai , India
pravin_ane@rediffmail.com

Dr. S. Srinivasan
Professor and Head, Department Of Computer Science and Engineering, Anna University ,
Regional Centre, Madurai , India
sriniss@yahoo.com

Abstract

A Radial basis function (RBF) neural network based fault localization technique is proposed in this paper to assist programmers in locating bugs effectively. Here we employ a three-layered feed forward artificial neural network with a radial basis function for its hidden unit activation and for linear function with its output layer activation. Here the neural network is trained to have a good relationship between the statement coverage information of a test case and its corresponding execution result to get a success or failure. The trained network is then given as an input to a set of virtual test cases, each covering a single statement, and the output of the network, for each virtual test case, is considered to be the suspiciousness of the corresponding covered statement. A statement with a higher suspiciousness has a higher likelihood of contain a bug, and thus, statement can be ranked in descending order of their suspiciousness. The Ranking can then be examined one by one, starting from the top, until a bug is located. Six case studies on different programs were conduced, with each faulty version contain a distinct bug, and the result clearly show that our proposed technique is much more effective than Tarantula, another popular fault localization technique.

***Keywords*: RBF neural network, Fault localization, ranking.**

1. Introduction

Regardless of how much effort goes into developing a computer program. It will still contain bugs. In fact the larger and complex a program, the higher the likelihood of it containing bugs. But to remove bugs from a program, we must first be able to identify exactly where they are. Knows as fault localization, this can be extremely tedious and time consuming, and is recognized to be one of the most expensive activities in program debugging. This has sparked the development of several fault localization technique, over the recent year, the main aim to assist developers in finding bugs, thereby reducing the manual effort spent. In this paper we propose an RBF neural network based fault localization technique that is more effective at locating bugs, in that a relatively smaller amount of code needs too be examined to find bugs, than other state of the art contemporary technique.

The neural network based on models has several advantages over other comparable models, such as their ability to learn. Given a sample dataset, a neural network can learn rules from the data with or without supervision. Neural networks are also fault tolerant by virtue of the fact that the information is distributed among the weights on the connections, and so a few faults in the network have relatively less impact on the model. In addition they have the capability to adapt their synaptic weights to changes in the surrounding environment. That is, a neural network trained to operate in a specific environment can be easily re-trained to deal with miner changes in the operating environmental conditions. Such qualities make neural networks popular among researches, and therefore, the neural networks have been successfully applied to many fields, such as pattern recognition, system identification, intelligent control, and cost estimation, reliability estimation and reusability. However in order to the best of our knowledge they have not been applied to help developers find bugs except for in our previous study, which uses a back propagation (BP) neural network based technique for fault localization .In this paper, we propose to use an RBF neural network based fault localization technique because RBF networks have several advantages of over BP networks, including a faster learning rate and a resistance to problems such as paralysis and minima.

A RBF neural network has a three-layer feed forward structure that can be trained to learn an input-output relationship based on a data set. In this Paper, the input is the statement coverage of a test case which indicates how the program is executed by the test case, and the output is the result (success or failure) of the corresponding program execution. Once the network has been trained, the coverage of a virtual test case with

only one statement covered is used as an input to compute the suspiciousness of the corresponding statement in terms of its likelihood of containing bugs. The larger the value of the output, the more suspicious the statement seems. Statements can be ranked in descending order of their suspiciousness, such as that developers can examine the ranking of statement (Starting from the top), one by one, until the first faulty statement is identified. Good fault localization technique should rank faulty statement towards the top, of their Rankings. An assumption that is typically made is that developers can correctly identify a faulty statement as faulty upon examination, and by the same token they will not identify a non- faulty statement as faulty. We systematically evaluate the RBF neural network technique across several different sets of programs each containing of many faulty versions. Result show that our proposed technique is much more effective than techniques such as Tarantula.

2. Related Works

The Debugging software is an expensive and mostly manual process. Of all debugging activities, locating the faults, or fault localization, is the most expensive. This expense occurs in both the time and the cost required finding the fault. Because of this high cost, any improvement in the process of finding faults can greatly decrease the cost of debugging. In practice, software developers locate faults in their programs using a highly involved, manual process. This process usually begins when the developers run the program with a test case (or test suite) and observe failures in the program. The developers then choose a particular failed test case to run, and iteratively place breakpoints using a symbolic debugger, observe the state until an erroneous state is reached, and backtrack until the fault is found. This process can be quite time-consuming.

To reduce the time required locating faults, and thus the expense of debugging, researchers have investigated ways of helping to automate this process of searching for faults. Some existing techniques use coverage information provided by test suites to compute likely faulty statements. Other techniques perform a binary search of the memory state using one failing test case and one passing test case to find likely faulty statements. Still other techniques are based on the remote monitoring and statistical sampling of programs after they are deployed. Most papers reporting these techniques also report empirical studies that evaluate the presented technique in terms of its effectiveness and efficiency. However, because of the difficulty in comparing techniques developed on different platforms for different languages and using different programs and test suites, few empirical studies have reported comparison of existing techniques.

3. Perspective Of The Proposed Work

The proposed architecture of the neural network along with their Ranking process is considered. The proposed approach uses a new process in statement can be ranked.

3.1.Proposed Architecture

The main aim of the Software Testing is of to find as many numbers of faults as soon as possible. Here we propose a virtual text case mechanism with a reduced test suite selection by selecting a subset from the original test suite and covered statements .this statements are consider in rank process. A test case in software engineering is a set of conditions or variables under which a tester will determine whether an application or software system is working correctly or not.

Test case Prioritization: is of selecting test cases in an order of higher priority with an earlier execution and has components that describe an input, action or event and an expected response, to determine if a feature of an application is working correctly.

- Prioritization methods
  - Initial ordering
  - Reverse ordering
  - Random ordering
  - Based on fault detection ability

Test suite reduction: In software development, a test suite, less commonly known as a validation suite, is a collection of test cases that are intended to be used to test a software program to show that it has some specified set of behaviors. A test suite often contains detailed instructions or goals for each collection of test cases and information on the system configuration to be used during testing. A group of test cases may also contain prerequisite states or steps, and descriptions of the following tests.

Test case selection: The main aim is to improve the effectiveness of regression testing by using test case prioritization techniques in order to execute the most beneficial. The test case selection is done by a new proposed Data Mining algorithm. Also we present results from an empirical study that compared the performance of different existing search algorithms applied to certain test cases and proposing a new algorithm.

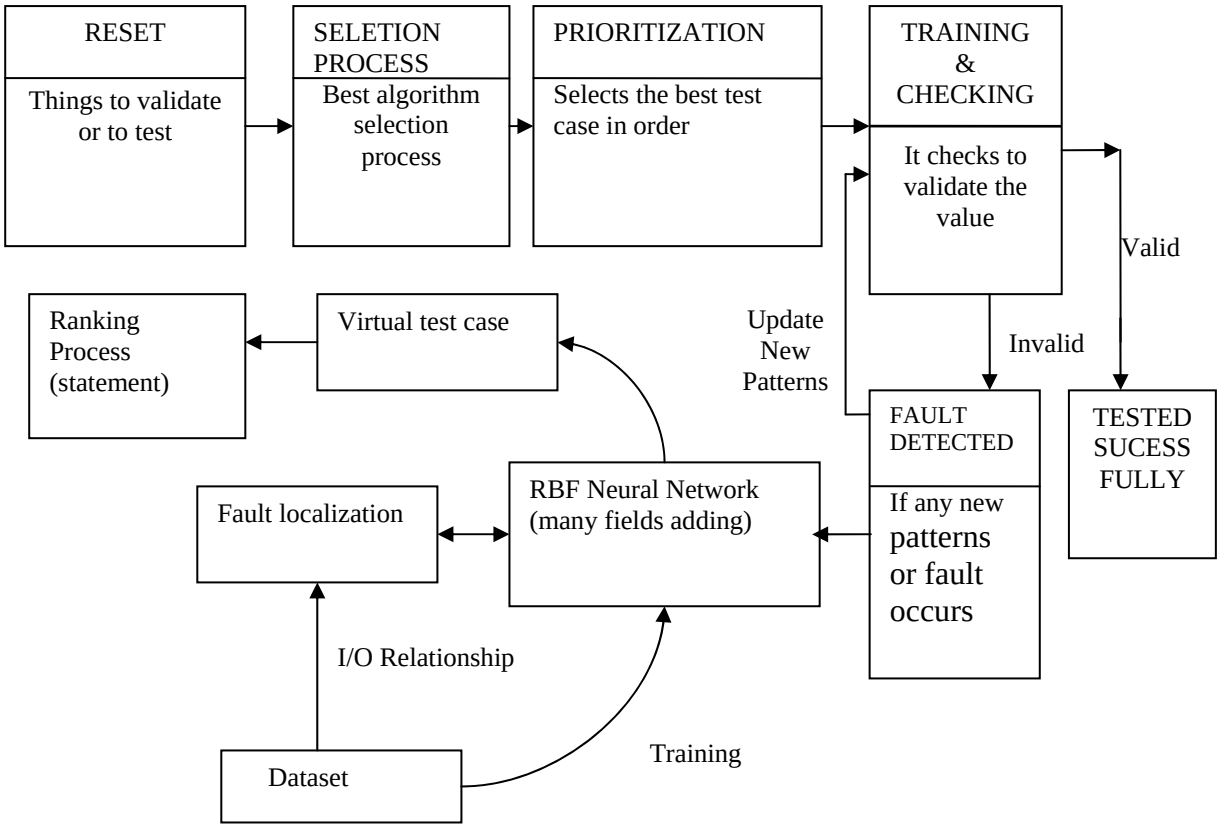

Fig.1: Proposed Model Architecture

3.2.Fault Localization technique and neural network

We present an overview of the five fault localization techniques we studied Tarantula, Set union, Set intersection, Nearest Neighbor, and Cause Transitions. For each technique, we describe the first, its method for computing an initial set of suspicious statements in the program| the set of statements where the search for the fault should begin and second its method of ordering (or ranking) the rest of the statements for containing the search in case the fault is not found in this initial set of suspicious statements. The RBF neural network has a three-layer feed forward structure that can be trained to learn an input-output relationship based on a data set. The input is the statement coverage of a test case which indicates how the program is executed by the test case, and the output is the result (success or failure) of the corresponding program execution. Once the network has been trained, the coverage of a virtual test case with only one statement covered is used as an input to compute the suspiciousness of the corresponding statement in terms of its likelihood of containing bugs.

The high cost of locating faults in programs has motivated the development of techniques that assist in fault localization by automating part of the process of searching for faults. Empirical studies that compare these techniques have reported the relative effectiveness of four existing techniques on a set of subjects. These studies compare the rankings that the techniques compute for statements in the subject programs and the effectiveness of these rankings in locating the faults. However, it is unknown how these four techniques compare with Tarantula, another existing fault-localization technique, although this technique also provides a way to rank statements in terms of their suspiciousness. Thus, we performed a study to compare the Tarantula technique with the four techniques previously compared. This paper presents our study it overviews the Tarantula technique along with the four other techniques studied, describes our experiment, and reports and discusses the results. Our studies show that, on the same set of subjects, the Tarantula technique consistently outperforms the other four techniques in terms of effectiveness in fault localization, and is comparable in efficiency to the least expensive of the other four techniques.

3.3.Algorithm Pseudocode for the proposed Approach

1. Let D be a damping factor which can be data set between 0 and 1.
2. Let V be a validation set.
3. Let T be a training data.

SRank algorithm is in fact elegantly simple and is calculated as follows:

$$SR(A) = (1-d) + d\,(SR(T_1)/C(T_1) + \ldots + SR(T_n)/C(T_n))$$

Where SR (A) is the searchRank of a statement A

4. SR← is the searchRank
5. C← is the number of outgoing links.
6. d is a damping factor in the range $0 < d < 1$, (binary values)
7. The virtual test case is covered with only one statement after it sends the information to the Ranking process.
8. Statements that can be ranked in descending order.

First of all, the Search algorithm Rank does not rank the test cases as a whole, but is determined for each test case individually. Further, the rank of statement A is recursively defined by the Search Rank of those statements which link to statement A. The Search Rank of statement Ti which link to statement A and does not influence the Search Rank of statement A uniformly. The Rank algorithm searches a Rank of statement T in which the weights the number of outbound links C (T) on statement T. This means that more number of outbound links of the statement T which has a lesser statement A with a benefit of ink in the statement T.

4. Results and Discussion

The results shown in Figure 2(a) depict the Number of Program Statement tested Vs percentage of the test programs that are been executed successfully. Also the Figure 2(b) explains about the Number of Program Statement tested Vs Time period that are taken for testing the programs in Hours

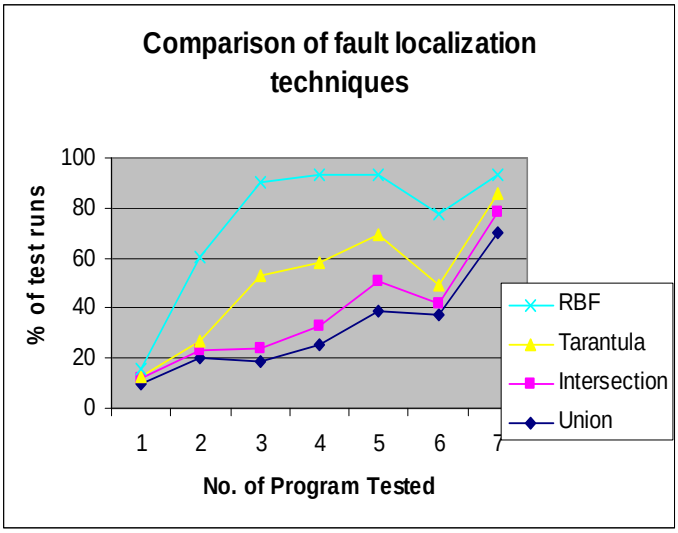

Fig2(a)Program Statement tested Vs Test run

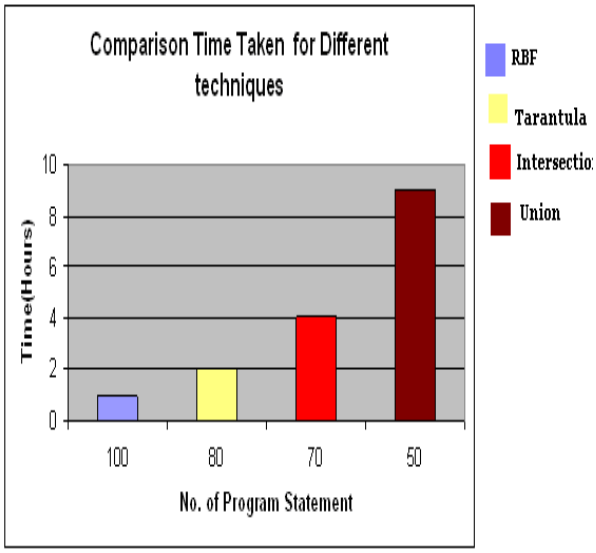

Fig2(b)Program Statement tested Vs Time Taken

The horizontal axis represents the score that are been measured and which represents the percentage of the subject program that would not be examined when the following orders of program points specified by the technique. The vertical axis represents the percentage of test runs that are found at the score given on the horizontal axis. For the RBF technique there is one test suite used for each faulty version, so the horizontal axis represents not only the percentage of test runs, but also the percentage of versions. For the Set-intersection, Set-union multiple test cases are chosen for each version. For these the vertical axis represents the percentage of all version-test pairs.

4.1.Comparison Of RBF with Tarantula

Figure 3(a) gives the effectiveness of the RBF and Tarantula technique for all programs used in our case study .The curves labeled as RBFBest (in red) and RBFWorst (in blue) are for the best and the worst case effectiveness when testing the RBF technique in different cases and conditions, and those labeled (in black) are Tarantula TWorst and in green) are for the best. Since these curves are displayed in different colors and they are identified and classified for different cases and conditions. For a given x value the percentage of the Executable statement that are been examined and its corresponding y value is the percentage of the faulty versions that are been occurred.

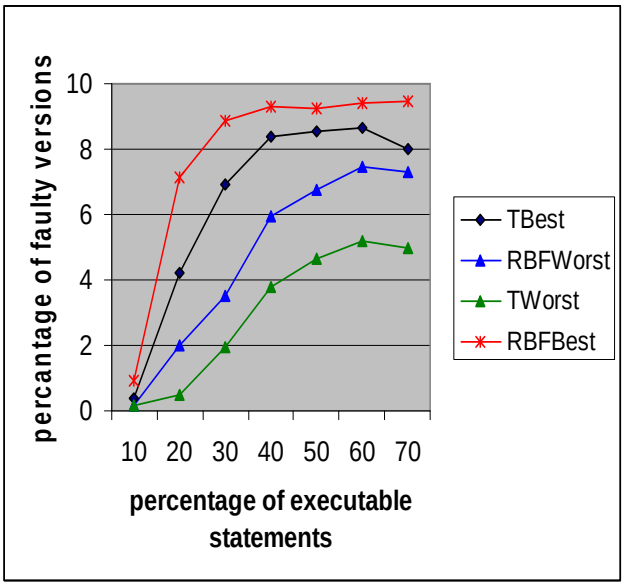

Fig.3: Executable statement examined Vs Faulty Percentage

5. Conclusion

An RBF neural network-based fault detection and localization technique is presented in this paper. The network is trained with coverage of information for each test case paired with its execution result for either success or failure. Also it is trained by giving input to several set of virtual test cases, each of which covers a single statement. The output of the network is considered to be of suspiciousness statement corresponding to the virtual test. Also the Statements with a higher suspiciousness is examined first, as they are more likely to contain program bugs and thus RBF is significantly more efficient in fault localization. Also the RBF fault localization technique is an efficient on ongoing process technique for a large-scale, software development environment.

References

[1] Fault Prediction in Object-Oriented Software Using Neural Network TechniquesAtchara Mahaweerawat, Peraphon Sophatsathit., Chidchanok Lursinsap. and Petr Musilek .
[2] Generating test data from state-based specifications Jeff Offutt1, Shaoying Liu2, Aynur Abdurazik1 and Paul Ammann1.
[3] Fault Localization using Execution Slices and Dataflow Tests Hiralal Agrawal, Joseph R. Horgan, Saul London, and W. Eric Wong ffhira, jrh, saul.
[4] On the Use of Neural Networks to Guide Software Testing Activities Charles Anderson Anneliese von Mayrhauser Rick Mraz.
[5] Mohammed Fraiwan "A survey of fault localization techniques in computer networks".
[6] Josh Wilkerson:" Automated Fitness Guided Fault Localization**"**
[7] H. Cleve and A. Zeller. Locating causes of program failures.
[8] H. Pan, R. A. DeMillo, and E. H. Safford. Failure and fault analysis for software debugging .

Authors

A.Pravin received the B.E degree in Computer Science & Engineering from Bharath Niketan Engineering College, Madurai Kamaraj University, Madurai, India in 2003 and M.E degree in Computer Science & Engineering from Sathyabama University, Chennai, India in 2005 .Where he is currently working towards the Ph.D degree in Computer Science & Engineering at Sathyabama University, Chennai, India.

He works currently as an Assistant Professor for the Department of Computer Science and Engineering at SRR Engineering College, chennai and he has more than 7 Years of teaching experience.He has participated and presented many Research Papers in International and National Conferences. His area of interests includes Software Engineering, Data mining and Data warehouse.

Dr. S. Srinivasan received the Ph.D degree in Computer Science & Engineering from the Sathyabama University, Chennai, and M.Tech. in Computer Science & Engineering from the Indian Institute of Technology, Chennai, and M.B.A. in Systems & Finance from Sathyabama Engineering College, University of Madras, Chennai, and the M.Sc. in Mathematics from the Gobi Arts College, Gobichettipalayam, Bharathiar University, Coimbatore. He is presently working as Professor and Head, Department of Computer Science and Engineering , Anna University, Regional Centre, Madurai. He has 20 Years of experience in Teaching and Research. He has also held various positions and responsibilities in Technical Institutions. He is acting as expert member at various universities in various capacities.. He has published more than 40 research papers in journals, books, conferences, and workshops. His research interest include text mining, data mining & data warehouse, and Software Engineering.