FP- International Journal of Computer Science Research (IJCSR)
Volume 1 Issue 1, Pages 36-44, 2014

Analysis and Design of Test Case Prioritization: Using Neural Networks and Classifiers

Prof. (Dr.) Amit verma
Head Department of Computer Science & Engg.
UIE, Chandigarh University, Ghuaran
amit.verma@cumail.in

Simranjeet Kaur
Department of Comp. Science & Engg.
Chandigarh University, Ghuaran.
simranjeetgill23@gmail.com

Abstract- In software testing, due to large suit size scheduling of test cases is important by using effective test case prioritization technique so that the test cases having higher priority are executed before lower priority. The objective of test case prioritization is earlier fault detection. In this paper code based and model based test case prioritization algorithms are defined which suggests that it is effective to prioritize the test cases on behavior basis as more faults are detected when different components interacts with each other, the analysis of various algorithms shows that Genetic Algorithm performs well, although Greedy approaches are effective in code based prioritization, for model based Prim's algorithm gives better result. In order to automate the test case prioritizations neural network is used. For reducing complexity and increasing speed with robust efficiency the requirement of classifier will play an important role.

Index Terms- Software engineering, software testing, test case prioritization, test case prioritization algorithms, regression testing, neural networks, classifiers.

1. Introduction:

Software engineering is the study and application of engineering to the design, development and maintenance of the software. Software engineering first conference held in 1968 in which Fritz Bauer defined software engineering as " The establishment and use of sound engineering principles in order to obtain economically developed software that is reliable and work efficiently on real machines". Over the past few decades, the use of digital computers diversified and increased with the corresponding increase in the cost of the software failure, the importance of testing grew. Various techniques are implemented to make the software testing process cost effective as advancements are increasing in this field.

The next section of this paper precisely describes the introduction about test case prioritization, evolution of software testing techniques, section 2 describes the classification of prioritization techniques and algorithms, and section 3 describes the neural networks in software testing process and section 4 concludes the paper and section 5 represents the future work.

.1.1 Evolution of Various software testing techniques:

Phase I: Before 1956:-

In 1950s it is debugging oriented period. In this phase software testing is defined as "What programmers did to find the bugs in the software".

Phase II: 1957-78:-

After 1957 it is demonstration oriented period in which testing is done to make sure that the software meets its specification or not. Until 1957 testing was known as program checkouts but it was distinguished from debugging.

Phase III: 1979-82:-

This period is Destruction oriented period in which testing is done to detect the implementation faults. In 1980 the definition of testing extended to include defect prevention which is done by designing the test cases.

.**Phase IV: 1983-1987:-**

In mid 1980's there is migration period from programming in small concepts to programming in large ,which affects the software development and leads to increase in complexity which enhances the evolution of various different testing strategies

Phase V: 1988-2000

In 1991, all the test activities are integrated using software development life cycle.

In 2000 all testing procedures are done based on UML specifications.

Phase VI: 2001-2005

In 2001 the main emphasis continuous on UML based specification. In 2002 exploratory testing is followed. In 2004 testing is done based on scenarios.

Phase VII: 2006-2014

From 2006 main focus is on Automatic test input generation, model based testing, symbolic execution, regression testing, search based testing. In 2013 Rational Unified Process is an iterative framework for software development. RUP is a specific implementation of the unified process.

1.2 Test Case Prioritization: The test case prioritization plays very significant role in software testing. The test case prioritization problem is defined (by Rothermel et.al.) as follows:

Definition 1.1 (The Test Case Prioritization problem): Given a test suite T, the set of permutations of T, PT, and function f that assigns an award value to an ordering of T, find T'∈PT such that:

$$(\forall T(T \in PT)(T \neq T')[f(T') \geq (T)] \quad ...(1)$$

Here, the function f is the quality measures of T applied to any such ordering which yields an award value for that ordering; PT represents the set of all prioritizations (orderings).

TCP Techniques use the entire test suite and decrease testing cost by achieving parallelization of the debugging and testing activities [18].

2. Classification of Test case Prioritization:

1. Code Based TCP: The code-based TCP uses source code of the software system to prioritize the test cases.

2. Model Based TCP: The model-based TCP uses the model which represents the desired behavior of the software system to prioritize the test cases.

2.1 Code-Based Test Case Prioritization Algorithms:

Search Based Test case prioritization algorithms are two types. The Meta heuristic algorithm and Greedy algorithm. The Meta

heuristic algorithm includes two algorithm i.e. Hill Climbing and Genetic algorithm. The Greedy algorithm includes greedy, additional and 2-optimal greedy algorithm.

2.1.1Greedy Algorithm:

Greedy algorithm is based on the algorithms that make decisions at some particular criterion. Greedy algorithms fails to find the global optimum solution to any problem .A Greedy algorithm is an implementation of "the next best" search philosophy [7].This algorithm works on the principle in which the test case having the maximum weight is selected first, followed by the test case with the second-highest weight until all test cases are covered with sub-optimal solution.

Coverage may be in terms of requirements; fault detection etc depends upon the tester opinion. Greedy search reduces cost of implementation, reduces execution time and results produced are of high quality.

Consider an example in Table 1, Test case A is selected first because it covers six statements which is maximum coverage. Test case B, which covers five statements, is selected next. Test case C and D cover the same number of statements and so the Greedy Algorithm either returns A,B,C,D or A,B,D,C .For E.g., If the both the test cases satisfies the same requirements then we can select them randomly.

Test Case	Statement							
	1	2	3	4	5	6	7	8
A	X	X	X			X	X	X
B	X	X	X				X	X
C	X	X	X	X				
D					X	X	X	X

Table1: A case in which the Greedy Algorithm Will Not Produce an Optimal Solution.

Greedy Algorithm:

Requirement: Initialize an empty network G containing n nodes.

Algo_Greedy

Begin
Evaluate the Weight of every test case n in network G.
Evaluate the score of G.
Initialize current item==0;
Sort (item List); //According to the coverage using Quick sort)
For (i=0; i≤ n;i++)
{
If(Weight of current item1>item)
Then
Assign current Item1== item;
}
Else if
{
Repeat Step6(until all the test cases weight is sorted in decreasing order)
}
Evaluate Step 10
End.

2.1.2Additional Greedy Algorithm:

The logic behind the working of additional greedy algorithm is same as total greedy algorithm, but little different strategy is followed as they depends upon the feedback received from the previous selection of test cases. Sihan Li.et.al. [4] Describes the main difference between the Total greedy algorithm and the additional greedy algorithm is that the additional greedy algorithm selects the test cases that satisfy the most not-yet covered requirements.

For the example in Table1,There are total 8 requirements or statements to be covered but test case A covers only statement 1,2,3,5,7and 8 . Test case B also not covers the statement 4 and 5 so we would not prefer test case B to be executed after A according to the definition of Additional greedy algorithm. As test case C and D both covers these statements so higher priority is given to them .Thus, Additional Greedy would return either A, C, D, B or A, D, C, B. After 100% is achieved then remaining test cases can be prioritized using any algorithm.

2.1.3 2-Optimal Algorithm:

The 2-Optimal Greedy Algorithm is an object (instance) of the K-Optimal Greedy Approach [9][7] when K=2. This approach is used in the area of heuristic search to solve the travelling Salesman Problem (TSP). The requirements which are yet not satisfied are fulfilled by 2-Optimal Greedy algorithm by selecting the two test cases which collectively satisfied the requirements by considering the internal order. Instead of achieving global optimum solution, suboptimal solution is achieved. The 2-Optimal approach has been found to be fast and effective [9].

For the example in Table1, test case C and D are chosen first since the combinations of test cases C and D covers eight statements and this is the maximum among all pairs of test cases and this approach achieves the global optimum for this example.

2.1.4 Hill Climbing:

Hill Climbing is part of well known local search algorithm. By finding a neighbor fitter it iteratively improves the solution by initiating the work with random solution. It terminates when no further improvement is achieved [4] or until the termination condition executes. This strategy has two primary variations: steepest ascent and next best ascent.

The steepest ascent Hill climbing algorithm for test case prioritization is consists of following steps:

1. Pick a solution state randomly and assign it as initial state.
2. Current state neighbors are evaluated.
3. Large fitness state is evaluated and move to that state from current state only if neighbor state has large fitness then current state otherwise no move is made.
4. Previous steps are repeated only if the current state remains unchanged.
5. Return the current state as the solution state of the problem.

Algo_Hill Climbing

Randomly select current Item in the search space;
Until n Iterations ≤ Max Iterations do
neighbors = getNeighbours(Current Item);
next Item=get Best Neighbors (neighbors);
If Evaluate (next Item) ≥ Evaluate (Current Item) then
Current Item = next Item;
End.

2.3 Genetic Algorithm:

Genetic Algorithm is based on the processes of genetic selection according to Darwin theory of evolution. The GA procedure proceeds with initializing a population P, evaluating individuals, and selecting pairs of individuals that are combined and mutated to generate new individuals, forming the next generation.

```
Algo_Genetic
Begin
t ← 0;
initialize P(t)
evaluate P(t)
While(not termination condition) do
Begin
t ← t + 1
select P(t) from P(t-1) according to evaluation
crossover P(t) according to crossover rate
mutate P(t) according to mutation rate
evaluate P(t)
 end
```

2.2 Model Based Test Case Prioritization:

Model based prioritization includes prioritizing the test cases on the basis of system-user interaction scenario basis. Event sequence graphs differ from the other modeling techniques, as they represent the behavior of the system when interacts with the user expectations.

2.2.1 Prim's Algorithm:

Prim's algorithm detects the fault effectively and with limited resources. This algorithm is used in graph theory where it finds the minimal spanning tree have all the edges connected with each other. The main aim of this algorithm to find the subset of edges which covers all the vertices but not necessary all the edges to be included.Prof.AS Adiga et.al [20] proposed system component interaction graph (CIG) in which interstate and intrastate components are categorized. The higher priority is given to intrastate components and lower priority is given to interstate components.

The more importance is given to the component interactions as many defects arise when they interact with each other. Prim's algorithm minimizes the cost of system retesting and maximizing the cost of objective function.

TS = T1, T2….Tj, Test suite.

C1, C2, C3…….Cn, set of components.

S1, S2, S3……Sn, set of states.

U= Time

```
Algo_Prim's
Begin
Initialize T {}
Set[intracount]=0,
Set[intercount]=0;
   If
   {
   Ck(Si)->Ck(Sj)
   then w=0
   }
   else if
   Ck(Si)->Ck(Sj)
   then w=1
   else
select appropriate value for  Ck(Si) and
Ck(Sj)

initialize
Set S={s}
       For
       {
        (T1=S(T),T1<Lw , T1++)
 // Add test case suite of state into T1
        }
  If
  w=0;
```

```
        for
        {
        (T1=S(T),T1<LW , T1++)
        }
Else
       {
        For (T1=S(T),T1<LW , T1+n)
        }
N is test case suite of state into T2
End if;
      V/S! = ∅
   then find lightweight interactive state
//If no more state are connected.
// T= T+T1; // Add T1 into T
// T= T+T2; // Add T2 into T
// Set T'= Reverse of T
//Output T'
End
End
```

3. Using a Neural Network in the Software Testing Process:

Today's scenario due to the increase in complexity of software programs, automated software testing methods is needed. Artificial Neural Networks are used in software testing to handle different aspects of the software testing. According to the specifications, by using the randomly generated data a multi-layer neural network is trained on the original software application. The trained neural network is simulated model of the neural network [3].

Artificial Neural Networks:

Design of Artificial neural network resemble with the structure and information – processing capabilities of the brain. The superior computational ability of the neural network is due to its parallel distributed design and its capability to extrapolate the learned information to yield outputs for inputs not presented during training.

The Back propagation neural network used is capable of generalizing the training data. As the data doesn't have the uniform representation so the neural network doesn't applied to the raw data. For Training purpose, the unit with higher output value is considered to be "winner". The initial synaptic weights of the neural network are obtained randomly and covered a range between -0.5 and 0.5. The neural network is best method of testing a software application if all the coverage conditions are satisfied.

Artificial Neural networks are analyzed to have potential to perform well for classification problems in many different environments, including business, science and engineering[11].

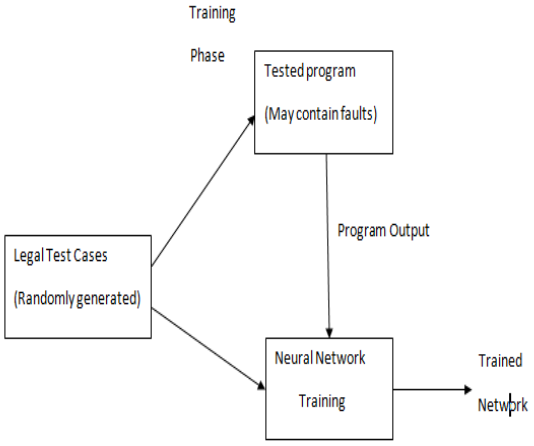

Fig 1: Overview of the training phase

Classification: Classification is most important in neural networks. Feed forward back –propagation (FFBP) learning

algorithm is commonly used learning algorithm for neural network classification. The Inaccuracy of NNs is reduced using gradient-descent search method to adjust the connections. Multi-layer perceptrons are a popular form of feed-forward neural networks with many successful applications in data classification [18]. Levenberg-Marquardt algorithm is best training algorithm in feed forward neural networks.

Mubariz Eminli[18] explain the classification –Based test case prioritization in which test cases and complete event sequences are ranked according to preference degree by Eq(2)

$$\text{PrefD(CESq)}=\sum_{i=1}^{n} Imp(xi)\mu sk(xi)fq(xi) \quad \text{-(2)}$$

Where μ_{sk} (xi) is the membership degree of the i^{th} event belonging to the cluster S_k , and f_q (x_i) is the frequency of occurrence i^{th} within CES_q. The value of μ_{Sk} (x_i) ranges between the [0, 1] and {0, 1} in fuzzy clustering and hard clustering respectively. *Imp* (x_i) is the importance of i^{th} event belonging to k^{th} cluster is defined as:

$$Imp(xi) = c - ImpD(Sk) + 1 \ldots\ldots\ldots (3)$$

Where c is the cluster, *ImpD* (S_k) is the importance degree of the k^{th} cluster.

Highest to lowest priorities are assigned to different test cases by ranking the calculated *PrefD*(CES_q) values.

Mubariz Eminli et.al [18] improved the clustering-based approach by classifying the preference degrees of all existing test cases to obtain the priority group label of any new test cases instead of assigning priority to individual test cases as it is very time consuming process. In such a way all the test cases formed data set is classified by using MLP neural network.

4. Conclusion:

This paper described code-based and model-based test case prioritization techniques by which it is concluded that to prioritize the test cases on the basis of user interaction with the system is more efficient as more number of defects can be detected when the different components interacts. Five code-coverage and search based algorithms are studied for the sequencing problem in the test case prioritization. The data and analysis indicates that the Greedy Algorithms performs much worse than Additional Greedy, 2-Optimal, and Genetic Algorithms overall. On the basis of effectiveness there is no significant difference between the performance of the 2-Optimal and Additional Greedy algorithms. The previous line suggests that, where applicable, the cheaper-to-implement-and-execute Additional Greedy Algorithm should be used.

Test suite size determines the size of the search space, thereby affecting the complexity of the test case prioritization and computation of the fitness value. The analysis of fitness function shows that the situations in which the total coverage consideration is required, the Greedy Algorithm don't produce the appropriate results.

Automation of test case prioritization is required due to large test suites which can be done with the help of neural networks .In order to satisfy certain conditions along with prioritization of datasets, classifiers are required.

5. Future Work:

Design and development of test case prioritization algorithm to automate the test

case prioritization process by using neural networks and classifiers.

6. References:

[1] Guoqiang Peter Zhang, "Neural Networks for Classification: A Survey" IEEE Trans.On Man, Systems, and Cybernetics, vol.30, no.4, Nov 2000.

[2] G. Rothermel, R. Untch, C.Chu, M.J. Harrold, "Prioritizing Test Cases for Regression Testing," IEEE Trans. Software Eng.,vol.27,no.10,pp.929-948,oct.2001.

[3] Meenakshi Vanmali, Mark Last, Abraham Kandel, "Using a Neural Network in the Software Testing Process", International Journal of Intelligent Systems, vol.17, page no. 45-62, 2002

[4] Sebastian Elbaum, Alexey G. Malishevsky, G.Rothermel, "Test Case Prioritization: A Family of Empirical Studies", IEEE Trans. Software Eng., vol.28, no.2, Feb.2002.

.

[5] D. Richard Kuhn, Senior Member, Dolores R. Wallace(2004), "Software Fault Interactions and Implications for Software Testing, IEEE Trans. on software engineering, vol. 30, no. 6, June 2004.

[6] Erick Cantu-Paz, Chandrika Kamath, "An Empirical Comparison of Combinations of Evolutionary Algorithms and Neural Networks for Classification Problems", IEEE Trans. On Systems, Man, And Cybernetics, 2005.

[7] Sihan Li, Naiwen Bian, Zhenyu Chen*, Dongjiang You, Yuchen He, "Simulation Study on some search algorithms for Regression Test Case Prioritization".

[8] Lu Luo, "Software Testing Techniques", Class Report for 17-939A, Institute for Software Research International Carnegie Mellon University
Pittsburgh, PA15232 USA.

[9] Zheng Li, Mark Harman, and Robert M. Hierons, "Search Algorithms for Regression Test Case Prioritization", IEEE Trans. Software Eng., vol.33, no.4, April 2007.

[10] Usman Farooq, C. P. Lam, "Evolving the Quality of Model Based Test Suite", IEEE International conference on Software Testing verification and validation workshops, 2009.

[11] Randall S. Sexton and Robert E.Dorsey, "Reliable Classification Using Neural Networks: A Genetic Algorithm and Back propagation Comparison".

[12] Dr. Arvinder Kaur and Shivangi Goyal "A Bee Colony Optimization Algorithm for Fault Coverage Based Regression Test Suite Prioritization" International Journal of Advanced Science and Technology Vol. 29, April, 2011

[13] M. S. Geetha Devasena, M. L. Valarmathi, "Meta Heuristic Search Technique for Dynamic Test Case Generation" International Journal of Computer Applications (0975 – 8887) Volume 39– N.o.12, February 2012.

[14] A. Mohamed Shameem,N. Kanagavalli, " Dependency Detection for Regression Testing using Test Case Prioritization Techniques", International Journal of Computer Applications (0975 – 8887) Volume 65– No.14, March 2013.

[15] Shifa-e-Zehra Haidry, Tim Miller, "Using Dependency Structures for Prioritization of Functional Test Suites", IEEE Trans. Software Eng. vol.39, no.2, Feb.2013.

[16] Thillaikarasi Muthusamy, Dr.Seetharaman.K, "A Test Case Prioritization Method with Weight Factors in Regression testing Based on Measurement metrics", International Journal of Advanced Research in Computer science and Software Engineering,Vol.3,Issue 12,Dec.2013.

[17]Orso, Gregg Rothermel "Software Testing: A Research Travelogue (2000–2014)" ACM May 31–June 7, 2014, Hyderabad, India.

[18] Nida Gokce, Mubariz Eminli,"Model-Based Test Case Prioritization using Neural Network Classification", Computer Science & Engineering: An International Journal, vol.4, no.1, Feb.2014.

[19] Swapan Kumar Mondal.et.al and Dr.Hitesh, "Regression Test Cases Minimization for Object Oriented Programming using New Optimal Page Replacement Algorithm" Vol.8, No.6 (2014), pp.253-264.

[20] Shweta A. Joshi, Prof. D.S. Adiga, Prof. B.S. Tiple, "Effective Use of Prim's Algorithm for model based Test case Prioritization", International Journal of Computer Science and Information Technologies, Vol. 5 (3) , 2014, 3444-3447.

[21] Xiang Chen, Zhaofei Tan,Jian Xia,Pengfei He, "Optimizing Test Case Execution Schedule using Classifiers" ,Journal of software,vol.9,no.10,October 2014.