ISSN:0975-9646

Effective Use of Prim's Algorithm for Model Based Test case Prioritization

Shweta A. Joshi[#], Prof. D.S. Adiga[*], Prof. B.S. Tiple[#]

[#]*Master of Computer Engineering student,*
MAEER's MIT, University of Pune- 411038, Maharashtra, India
[*]*Associate Professor,*
MAEER's MIT, University of Pune- 411038, Maharashtra, India

***Abstract*— This Software testing is "Performing Verification and Validation of the Software Product" for its correctness and accuracy of working. Every time it is not possible to perform each and every test case. Hence it is important to decide test cases prioritization.**

The aim of Test case prioritization is to prioritize the test case sequences and finding the faults as early as possible to improve proficiency of testing. Here we define a new prioritization method which prioritizes test cases in descending order for Component Based Software Development (CBSD) using the concept of Prim's algorithm.

Our main aim is to implement and observe the model based test case prioritization algorithm which make use of CIG (Component interaction Graph) as input for a medium/ large size CBSD (Component Based Software Development Process) by taking any real-time system as an example and to generate prioritized test cases in descending order.

***Keywords*— CBSD, CIG, Prim's, Test Case Prioritization.**

I. INTRODUCTION

Testing includes distinguishing the test cases which can find the errors in the given code. Test case prioritization is a method to order the test case in a sequence that provides some objective function. Basically, Test case prioritization is classified in two methods: code-based and model-based test case prioritization. In code-based test case prioritization, source code of the system is used to prioritize the test cases. Whereas in model based method any type of model is used to prioritize the test cases. If test cases are prioritized with the help of model based method then fault finding is comparatively easy than code based method. Also a model-based test case prioritization may be a cheaper solution. Hence, a model-based test case prioritization is best approach for component based software system.

As there is rapid development in software industries there is a demand of increasing use of component based software development processes. A Component interaction graph depicts that interrelation among different components. Hence by representing component interaction using CIG we can outline the test case sequences.

Here we will discuss new test case prioritization technique which describes component interaction between different components by giving CIG as input and based on it which is able to prioritize test case sequences in descending order by applying Prim's algorithm on it.

II. PRIORITIZATION METHODOLOGY

As software time to market becomes shorter day by day, there is significant increase in the use of commercial off the shelf components. It is important to use the testing technique based on component which can increase the fault finding rate. There are different techniques to prioritize test cases which can detect more defects in lesser time. Following are some papers which list out different techniques of test case prioritization.

Majorly there are two different techniques; code based test case prioritization and model based test case prioritization. In code based method system is dependent on source code completely whereas in model based method system captures the structure and behavioural aspects. Also model based technique finds faults in very less time as compared to code based approach.

A.Model based test case prioritization using UML state chart diagram

In this technique test cases are prioritized for component based system retesting. Here state chart diagram is considered for each component which represents state transitions among each component then it constructs Component Interaction graph (CIG) to show an interrelation within components. Prioritization algorithm counts interrelation among components and interrelation within components. It depends on maximum state changes occur among and within components as well as database access during test case execution.

III. RELATED WORK

In today's world day by day drastic changes are in software which needs to be more proficient for this component based software development process provides platform to detect faults in quick time and refine code shortly. To perform such tasks developers/ testers needs to find out best one out of alternatives that is 'next best' known as 'greedy' method.

Basically greedy methods are simple, easy-to-implement and provide solutions to complex problems by deciding which next step will provide the most efficient result. Such algorithms are called greedy because optimal solution to each smaller instance will provide an effective output; the

algorithm doesn't consider the larger problem as a whole. Greedy methods works with two different criteria:

- *Kruskal's algorithm*

In this algorithm at each stage, the edge with the least cost is processed

Assume;

G = (V, E)

Keep track of connected components of graph with edges

Initially components are single nodes

At each stage, add the cheapest edge that connects two nodes not already connected

Experimental study of Kruskal's algorithm states that Kruskal can have better performance if the can be sorted in linear time; edges in each component are already sorted.

- *Prim's algorithm*

There are quite limitations with kruskal's method; so for our model based prioritization technique we finalized Prim's method to prioritize test cases which are generated in random sequence which we have to arrange in some sorted order.

A. Implementation Details of Prim's Algorithm

As discussed in model based prioritization testers need to check behavioural aspects of the system i.e. multiple state changes occurred, the test case is going to access the data base or single attribute or multiple attributes from a database access. Hence I am proposing new optimization technique i.e. Prim's Algorithm to explore more effective prioritized test cases which can find out defects in earlier time and can maximize the effectiveness of any medium/ large system. As already discussed Prim's algorithm is an algorithm in graph theory that finds a minimum spanning tree for a connected weighted graph. This means it finds a subset of the edges which converts it in a tree that includes every vertex, where the total weight of all the edges in the tree is minimized. If the graph is not connected, then it will only find a minimum spanning tree for one of the connected components.

Framework of model based test prioritization using prim's algorithm is given below in Fig. 1.

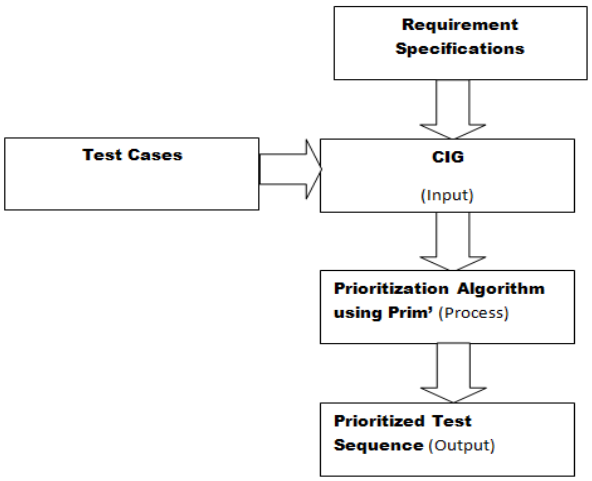

Fig. 1. Framework of proposed system

In this paper, for proposed system CIG graph defined as input data set along with number of test cases written for each state in components, direct access i.e. intrastate changes it means higher priority is given to intrastate change whereas, indirect access i.e. interstate changes which have less priority for implementing prioritized test case suite sequence . CIG of components shows the interaction between the components. Based on prim's strategy algorithm we follow the steps and prioritize the test cases. Also interstate change and outer state change for calculation to set the priority according to algorithm. Lastly we will show the decreasing order of test cases according to the priority. We also try to explore the graphical representation of time taken to execution if numbers of components are present.

B. Algorithm: Prioritization

Input:

TS=T1, T2…Tj, Test Suite.

C=C1, C2, C3...Cm set of components.

S= S1, S2, S3…S*n*, set of states.

U=time.

Output:

Prioritized test sequence T'

1. Start.
2. Set T' as Empty.
3. T1[intracount] = 0; T2[intercount] = 0;
4. If Ck (Si) -> Ck (Sj) Then w = 0. // If states are interacting in same component.
5. If Ck (Si) -> Cl(Sj) Then w = 1. // If states are interacting in different component.
6. Select any state of any component, set S = {s}.
7. T1 [intracount] = S(T) //Add Test Case Suite of State into T1.
8. intracount ++;
9. Find lightest weight interactive state such that one endpoint is in S and other is in V\S.
10. If (w = 0) Then
11. T1 [intracount] = S (T) //Add Test Case Suite of State into T1.
12. intracount ++;
13. Else
14. T2 [intercountt] = S(T) //Add Test Case Suite of State into T2.
15. intercount++;
16. End If
17. If (V\S! = ø) Then Step 9. //If no more state connected
18. T = T + T1; //Add T1 into the T
19. T = T+ T2; //Add T2 into the T
20. Set T' = Reverse of T.
21. Output T'

IV. RESULTS & DISCUSSION

This system is going to work with 5 modules named as

A.Requirement Specifications

In this module we have to count number of components and number of states in each component all data is stored and retrieved using MySQL.

B.Component Interaction Graph

With this module we can represent interactions and state changes in component or within different components i.e. interchanges and interchanges of states respectively using JavaAppletgraph we have shown here screenshot of CIG as below.

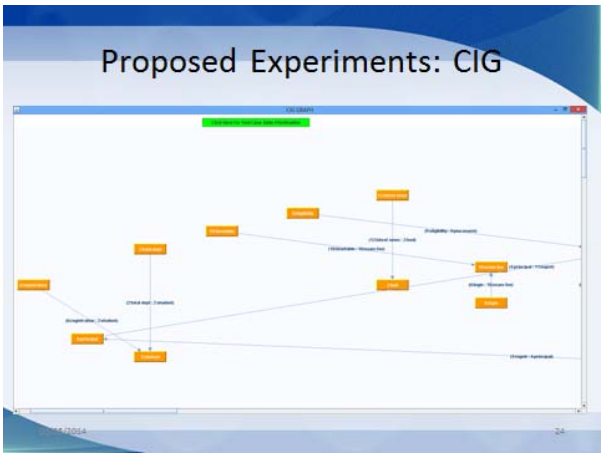

Fig 1 Screenshot of CIG

C.Test Cases

Here all test cases are generated for each state in the components and prioritization is defined with state changes in component or among components.

D.Prioritization using Prim's algorithm

In this module actual prioritization can be performed on the basis of CIG generated and number of test cases listed for every state with prior knowledge of Prim's algorithm. In following table first column represents more weighted test cases i.e. DB direct which are prioritized first and second column shows test cases which have less priority i.e.DBIndirect.

DBdirect and DBindirect

Table I
Result set 1 of test cases

DBdirect (intrastate changes)	**DBindirect (interstate changes)**
T1	T2
T2	T4
T3	
T4	

E. Prioritized test suites

After sorting test cases in this module we are finally representing prioritized test cases in decreasing order according to its priority as shown below

Prioritized Test Case Sequence

Table II
Result set 2 of prioritized test sequence

1.	T2
2.	T4
3.	T1
4.	T3

V. CONCLUSION

For prioritizing test cases there are many algorithm instead of these a new prioritization technique is introduced using prim's algorithm for finding defects in the component based software in less time. Here more importance is given to component interactions because maximum defect occur when components are going to interact with each other. This approach is mainly applicable to test the component composition in case of component based software maintenance.

Finally our algorithm is found to be very effective in

- Maximizing the objective function.
- Minimizing the cost of system retesting.

Future Challenges

In future we can extend this work as; the system can use regression testing strategy by adapting the knowledge of Genetic algorithm. Also we can compare efficiency of this algorithm with other searching algorithm.

VI. ACKNOWLEDGEMENT

I feel immense pleasure while presenting this work and am very thankful to my guide **Prof. *D. S. Adiga***, and ***Prof. B.S. Tiple***, Dept. of Computer Engineering of *Maharashtra Institute of Technology, Pune* for her valuable suggestions and support.

I would like to express my sincere thanks to our senior professor and M.E. Coordinator ***Prof. V.S.Jagtap*** for teaching me the fine points which are helpful for this completing work. I am also thankful to our Principal of MIT, Pune **Dr. L. *K. Kshirsagar*** for his consent to go forward with this topic.

REFERENCES

[1] Sanjukta Mohanty, Arup Abhinna Acharya, Durga Prasad Mohapatra, "A Model based prioritization technique for Component based software retesting using UML state chart diagram". 2011 IEEE Third Int'l Conf. on Electronics Computer Technology.

[2] G. Rothermel, R.H.Untch, C.Chu, M.J.Harrold, "Test Case Prioritization: An Emperical Study," in Proceedings of the 24th IEEE International Conference Software Maintenance (ICSM '1999) Oxford, U.K, September 1999.

[3] P. Malangave, D. B. Kulkarni, "Efficient Test case Prioritization in Regression Testing".

[4] Z.Li, M. Harman, and R. M. Hierons," Search Algorithms for Regression Test Case Prioritization," IEEE Transactions on Software Engineering, Vol. 33, No. 4, April 2007.
[5] B. Korel, G. Koutsogiannakis, "Experimental Comparsion of Code Based and Model model Based Test prioritization," IEEE 2009.
[6] Hema Srikanth and Laurie Williams, "Requirements-Based Test Case Prioritization," North Carolina State University, 2005, ACM SIGSOFT Software Engineering, pages 1-3.
[7] A. A. Acharya, S. K. Jena, "Component Interaction Graph: A new approach to test component composition," Journal of Computer Science and Engineering, 2001, Volume 1, Issue 1.
[8] Y. Wu, D. Pan and M. Chen, "Techniques for testing component-based software," In Proceedings of the 7th IEEE International Conference on Engineering of Complex Computer Systems, 001,Pp- 222-232.
[9] Sanjukta Mohanty, Arup Abhinna Acharya, Durga Prasad Mohapatra, "A survey on model based test case prioritization" International Journal of Computer Science and Information Technologies, 2011,Vol. 2 (3) , 1042-1047.
[10] G. Rothermel, R. Untch, M. Harrol, "Prioritizing Test Cases For Regression Testing," IEEE Transactions on Software Engineering, 2001,volume 27, No. 10, pp. 929-948.
[11] Arup Abhinna Acharya and Sisir Kumar Jena, "Component Interaction Graph: A new approach to test component composition" Vol 2(3), 2011, 1042-1047.
[12] P. R. Srivastava, "Test Case Prioritization," Journal of Theoretical and Applied Information Technology, 2008, JATIT.