Application of Anytime Interruptible Algorithm for Software Test Case Prioritization

Manas Kumar Yogi, D.Uma
Department of CSE, Pragati Engineering College, Surampalem, A.P., India
manas.yogi@gmail.com
umadarapu03@gmail.com

ABSTRACT

This paper describes the usage of the interruptible anytime algorithm for prioritization of software test cases to reduce test effort and time. We have combined the quality of output generated by decision trees to the software test cases. A software test case exercises the functionality of a software feature and many a times it becomes difficult to test all test cases within available testing time. So bypassing traditional way of test case prioritization strategies we have presented a novel mechanism where the greedy approach of decision tress is used on test metrics like customer assigned priority, requirements complexity, requirements volatility, average percentage of faults detected .As per studies, the customer assigned priority has the highest business impact so we have given utmost priority to this metric. The interruptible anytime algorithm can work in iterations also to improve the results. Using the algorithm presented in this paper we can save considerable amount of testing time.

KEYWORDS

Decision trees, Interruptible anytime algorithm, Prioritisation, Test case, Requirements

I. INTRODUCTION

Software test case prioritisation refers to arrangement of test cases in a specific order of importance while test case execution so that if time is less only the high priority test case will be executed. High priority test case is a test case which is robust and the result of this test case has high business value on the functionality of the software. High priority test case does not refer to the technical importance of the test case rather it is a qualitative indication of the verification of the feature of the software. So, this test cases are part of the acceptance criteria document too. Already huge work in software case prioritisation has been done but so far most of this approaches are static in nature. We say static as time has a important say in this methods. If testing time is uncertain then this approaches are not of high reliability. So we have proposed a unique algorithm of applying the interruptible anytime algorithm which works on unbound time factors and also ensures a high quality solution. We have applied this algorithm on a particular type of nonlinear data structure called as decision tress. Decision tress act as classifiers given the input with distinct features. It has high level of interpretability. It generates human understandable predictions when proper inputs are applied as input. We have applied input methods like values of test case prioritisation metrics based on our study of test case metrics which incur profitable margins when we have short time in hand. The technique we use can be improved based on factors which can be controlled given a specified degree of certainty. This has been discussed in section 4 of the paper.

II. PROPOSED TECHNIQUE

As per the interruptible anytime algorithm can be stopped instantaneously, the testing team need not worry about allocated time for testing. We use decision trees to model the test suite having T_i number of test cases.

We consider each test case depends on following metrics

a) Customer assigned priority (CAP)
b) Requirement complexity (RC)
c) Requirements volatility (RV)
d) Average percentage of fault detected (APFD)

We model the test cases as shown below.

From decision tree we will get prioritized order for any two test cases. Now we have to generalise for n test cases.

If n is even then we need maximize of n(n-1) comparisions,if n is odd we need maximize 2n comparisons. Consequently the number of comparisons to get an optimal test case prioritized order will be somewhere between these boundaries of n(n-1) and 2n.As decision tree uses a greedy approach, we can interrupt the growing phase of tree anytime without hassles disturbance of test effort time.

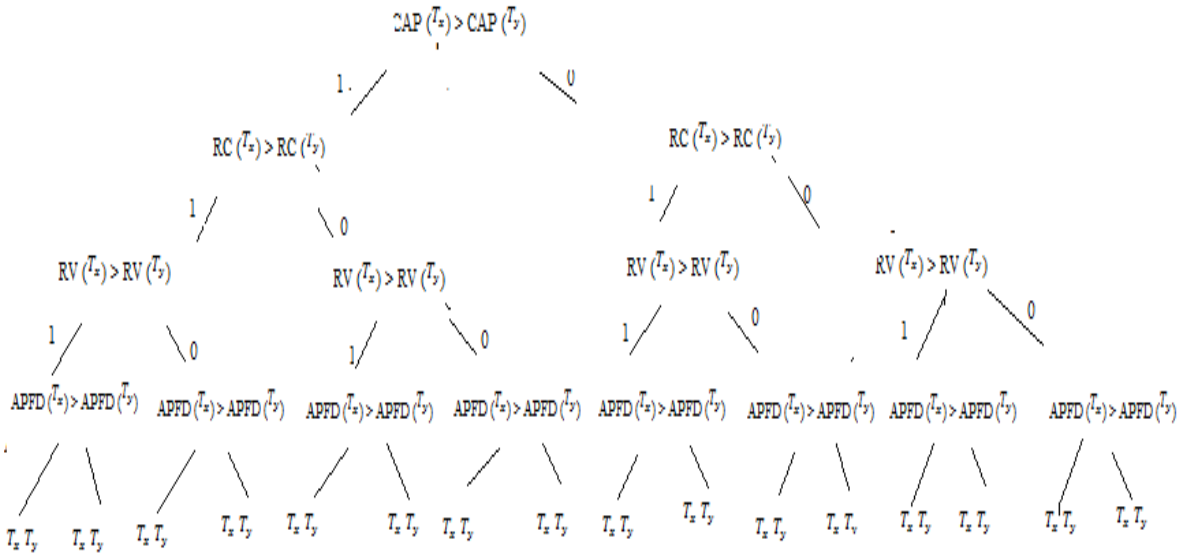

Figure 1. Decision tree for test cases T_x, T_y

III. GENERATION OF DESICION TREE

A Decision tree is a hierarchical structure where each node represents a decision. The result of taking a decision is either yes (or) no. The path where a Yes decision is taken is weighted as 1 and The path where a No decision is taken is weighted as 0.

We start generating a decision tree taking two test cases T_x , T_y belonging to a test suite T ,ie $\{T_x , T_y\}$ ε T. We need the prioritised order between T_x and T_y. If prioritised order is $\{ T_x , T_y \}$ it means we execute test case T_x followed by T_y else if prioritised order is $\{T_y , T_x\}$. We execute test case T_y followed by T_x .At the Root of the decision tree, We compare the values of Customer-assigned priority, i.e, CAP for T_x , T_y. We generate two branches depending on whether CAP (T_x) is more than CAP (T_y). At level 1, We compare the requirement complexity of T_x , T_y. We again branch out depending on RC(T_x) > RC(T_y) subsequently we model the level 2, level 3 of the decision tree based on requirement volatility and average percentage of Fault detected (APFD). We can observe from the

decision tree that the leaf nodes contain the test cases, i.e, either T_x , T_y .The possibilities of the prioritised order for the concerned test cases are as follows:
T_x followed by T_y
T_y followed by T_x
$T_x = T_y$ in this case we can randomly pick and test case as it won't affect the priority of test cases. We now proceed to apply the interruptible algorithm to the decision tree. As this algorithm stops the generation of the decision tree at any level say level 2 (or) even at level 1 the prioritised order is always a near approximation depending on the amount of computational resource allocated for the generation of the decision tree. The main goal for the testing team is to get the prioritised order with minimum resource allocation as well as turn around time. The interruptible algorithm also maintain the last executed result, so that if they are given more time, they can continue. From where they left off to procure a better result. We propose an algorithm in following manner.

Procedure IAATP (T_x , T_y)
T_p = CAP (T_x , T_y)
While not-interrupted
Node 1= CHOOSE_TNODE(T_p , RC(T_x) , RC(T_y))
Node 2= CHOOSE_TNODE(T_p , RV(T_x) , RV(T_y))
Node 3= CHOOSE_TNODE(T_p , APFD(T_x) , APFD(T_y))
Return T_p

In the above algorithm named IAATP refers to Interruptible anytime algorithm for test case prioritization. T_p will store the test case either T_x (or) T_y which has more priority.
CHOOSE_TNODE is a method which inputs 3 parameters. They are the current test case which has highest priority and the RC,RV,APFD values of T_x , T_y at each level of the decision-tree. We introduce 3 variables named Node 1, Node 2, Node 3 which store the test case with highest priority at level 1, level 2, and level 3 say at Node 1 T_x is prioritised at level 1,T_X again at level 2 gains priority and at level 3 will store T_x. Finally, We return $T_p = T_x$ as test case with highest priority.

IV. PERFORMANCE PROFILE FOR PROPOSED ALGORITHM

The algorithm we have proposed depends on uncertain factors also. One of this factor is that if at each level the RC, RV values are equal for Tx, Ty then we cannot move to level 3. In such a case we have to choose randomly from Tx, Ty. So, the main challenge is to specify distinct values of RC, RV for the test cases and input then to the proposed algorithm. So, if CAP for any number of test cases are equal, they are ruled out for the input of the algorithm. Only the test cases which have distinct CAP values are fed into the algorithm. The next difficult arises when CAP values are distinct but RC values are same. For this again we need to eliminate test case with same RC values. At level2, similarly we eliminate test cases with same RC values and subsequently we apply this principle in level 4 for test cases with same APFD values .But this approach takes time. So, we can allow the full decision. Free to generate even when two test cases have same values of RC, RV, APFD values. Yet another approach is that whenever we face such situations, we maintain separate memory list in which we go on string the test cases if their RC, RV, APFD values are equal. In this way the decision tree with generate leaves with distinct test cases only.

V. CONCLUSION

We conclude by advocating the application of the proposed algorithm in iterative way for all active test cases in a test suite. We intend to save the testing time and effort of the software testers by usage of this algorithm. In future we intend to apply the decision trees on generation of regression test cases. Test case prioritisation for regression test cases are difficult due to the nature of bug fixing work. Due to modifications done in the module we test only the test cases of modified modules as a result of which the test metric significance may change. If it does not change our algorithm is applicable but if changes then we need to reorganise the decision tree levels ie metric at each level will also change. Due to this cascading nature of test case metrics the applicability of the proposed algorithm for

prioritizing regression test cases may not be effective. Practitioners of regression testing should have a cautious approach while using our proposed algorithm.

REFERENCES

[1]. D. W. Opitz. An Anytime Approach to Connectionist Theory Refinement: Reasoning the Topologies of Knowledge-Based Neural Networks. PhD thesis, Department of Computer Sciences, University of Wisconsin-Madison, 1995.
[2]. Papagelis and D. Kalles. Breeding decision trees using evolutionary techniques. In ICML'01, pages 393-400, San Francisco, CA, USA, 2001.
[3]. J. R. Quinlan. Induction of decision trees. Machine Learning, 1:81{106}, 1986.
[4]. J. R. Quinlan. C4.5: Programs for Machine Learning. Morgan Kaufmann, San Mateo, CA, 1993.
[5]. S. J. Russell and E. Wefald. Principles of metareasoning. In KR'89, pages 400{411, San Mateo, California, 1989.
[6]. S. J. Russell and S. Zilberstein. Composing real-time systems. In IJCAI'91, pages 212{217, Sydney, Australia, 1991.