Application of Kraft–McMillan Inequality for Software Test Case Prioritization

Manas Kumar Yogi, Karri Vijaya Lakshmi, Koondrapu Koushik Sri Sai

Abstract**: *The motivation behind this prioritization is to improve the probability that if the experiments are utilized for relapse testing in the given request, they will more firmly meet some goal than they would on the off chance that they were executed in some unique request. A few associations want to run "Smoke" or "Sanity" test each time they get another form or form of the creating software. For this situation, experiments will be organize dependent on all the real modules of the software and sanity will be kept running on them to check the fundamental usefulness for instance, in a mobile testing, sanity test suite will have experiments like "restarting the gadget", "killing", "marking in", "refreshing software" etc. Whether the company runs relapse or sanity or both, Test Case Prioritization procedures are pertinent for every one of the cases. Organizing experiments should be possible based on necessities, expenses of bug fixing, history of the parent gadget and so forth. In this paper we apply a novel approach of data structures to develop a friendship relation between similar test cases so as to not spend time on testing similar functional test cases. At the end of the paper we find appreciable experimental results which outweigh the current techniques used in software testing area.***

Index Terms**: *Test case ,Prioritization, Kraft, McMillan, Friendship.***

I. INTRODUCTION

Testing is a crucial phase of software Development Life Cycle. Software testers prioritize the test cases which are more important, by some measure, are run earlier in the regression testing process. Regression testing is an costly testing procedure used to validate modified software. If we have thousands of test cases in regression suite and do not have sufficient time to execute all test cases, then we execute the test cases based on prioritization. Whatever the tests may be whether they are smoke, sanity or regression , Test case prioritization techniques are applicable for all the cases and used to schedule test cases. This is useful in order to minimize time, cost and effort during software testing phase. Testers can easily execute test cases, which have high priority and provide earlier defect faults. Some prioritization test cases may have similar test cases. Executing all the testcases results in waste of time. To eliminate similar test cases we establish a friendship relation between the trees constructed with test cases.

II. PROPOSED MECHANISM

Our paper proposes a property of Kraft–McMillan inequality to arrange the test cases according to a specific priority order . Kraft's inequality constraints the execution of test cases in a test suite: if we consider an complex and lengthy test suite of , that is, it must have total measure less than or equal to a healthy friendship value. Kraft's inequality can help in identifying test cases which may be redundant or similar and does not affect the quality of the testing process. We have defined a friendship value between 2 trees. Each of these binary trees store test case number which is related to a particular functionality. For each binary tree we apply the Kraft–McMillan inequality to obtain the value of

We Consider a set Ts ={T1,T2,T3……….T100} with 100 test cases and it is equally partitioned into two equal sets Ts1={T1,T2,T3,………..T50} and Ts2={T51,T52,T53,…..T100}with50 testcases in each set. Each set is partitioned into seven binary trees with equal number of testcases because the trees with different number of test cases may not have friendship. Test cases are arranged in a tree based on their prioritization and no testcase should be repeated. Ts11(7), Ts12(7), Ts13(7), Ts14(7), Ts15(7), Ts16(7), Ts17(8) are the trees obtained by dividing Ts1 Set. Ts21(7), Ts22(7), T23(7), Ts24(7), Ts25(7), Ts26(7), Ts27(8) are the trees obtained by dividing Ts2 set. we will calculate Kraft–McMillan inequality for each tree. Now, we will find the Kraft-McMillan difference between each tree in set1 to remaining trees in set2 to calculate the friendship between the trees. The difference is calculated as Kraft(Ts11)- Kraft(Ts21).similarly it is calculated for other trees. The Kraft–McMillan inequality difference is inversely proportional to friendship. If the friendship is between the range 0-0.05 then the two trees are said to have high friendship. .If the friendship is equal to 0.05 then the two trees are said to have medium friendship. .If the friendship is between the range 0.05-1.00 then the two trees are said to have low friendship. The trees with high friendship have similar test cases The total test cases in the two trees which have high friendship need not to be executed. The test cases either in tree1 or tree2 can be executed.

III. TYPES OF TEST CASES

A. Blockers

These are the test cases that test the life line of a software. The software can be useless if those are not performed.

Published By:
Blue Eyes Intelligence Engineering
& Sciences Publication

And it will block the other testing cases.

B. Critical

These test cases contain all the major functionalities performed by software.T hese functionalities are very important to users and if they fail then users will trash the software.

C. Relative dependencies

some test cases are executed sequentially that they can run only after others. They are dependent on others.

D. Timings of defect detection

Applies to cases where the new test cases can found only when the problems related to it are already fixed.

E. Major

These makes the software unique and different from other competitors. users will not be satisfied but may use the software as it have all basic requirements. This can lead to loss in business.

F. Difficulty levels

Testers prefers to execute simpler ones first rather than complicated ones.

G. Minor

These are the product improvements or small UI changes. They does not show any effect on software and these can be avoided if there is less time.

IV. CATEGORIZATION OF TEST CASES

Categories of test cases	**priority**	**Test cases id**
Blockers (lifeline-of software)	**1.1** Build Verification Tests	T_{26},T_{21},T_{78},T_{89},T_{49}, T_{100},T_{38},
	1.2 High risk effected module test cases	T_{84},T_4,T_{56},T_{74},T_{35},T_{43}
Critical(Major functionalities)	2	T_1,T_{50},T_{98},T_{67},T_{58}, T_{20},T_{46},T_{66},T_{73} ,T_{32}
User interface	3.1 Boundary value test cases	T_{99},T_{19},T_{81},T_{28},T_{65}, T_{60},T_{37},T_{40},T_{45},T_{63}
	3.2 Dependant test cases	T_3,T_{76},T_{13},T_{30},T_{95}, T_{80},T_{22},T_{69},T_{59},T_{34}
Database test cases	4	T_{87},T_{44},T_{23},T_{12},T_{92}, T_{14},T_{72},T_{53},T_{42}
security	5	T_{48},T_{94},T_5,T_{11},T_{36}, T_6,T_{88},T_{82},T_{61},T_{54}
New features(features added in new release)	6	T_{41},T_{24},T_{52},T_{75},T_{31}, T_{51},T_{93},T_{29},T_7,T,$_{83}T_{70}$
usability	7.1stress test cases	T_2,T_{25},T_{77},T_{57},T_{96},T_{15}, T_{85},T_{47},T_8,T_{62},T_{91},T_{55},T_{39}
	7.2 load test cases	T_{18},T_{27},T_{79},T_{97},T_{68},T_9, T_{16},T_{64},T_{33},T_{90},T_{71}
Minor(product improvements)	8	T_{17},T_{86},T_{10}

The trees are constructed based on the following conditions:

A tree must contain all types of test cases

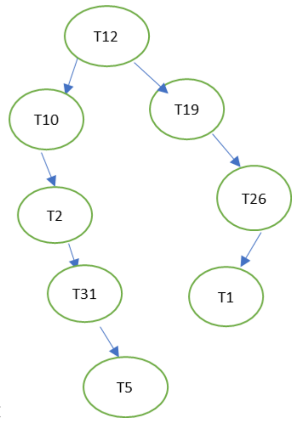
1.lowest priority test cases must be inserted to the left subtree

2.higest priority test cases must be inserted to the right subtree

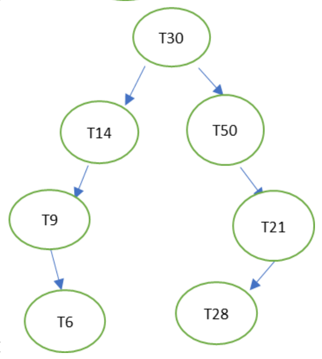
3.The medium test case should be the root

V. CONSTRUCTION OF TREES

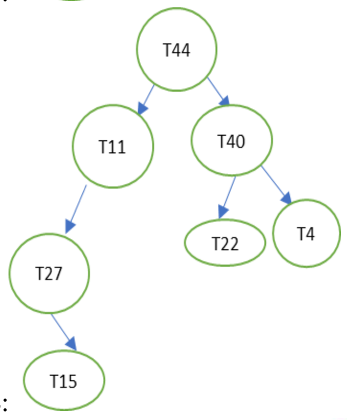
A. Trees constructed using T_{s1} set:



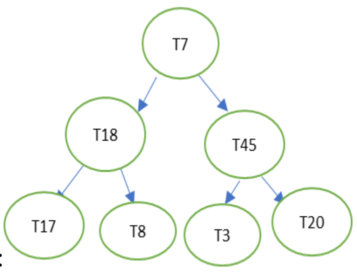
Ts11:



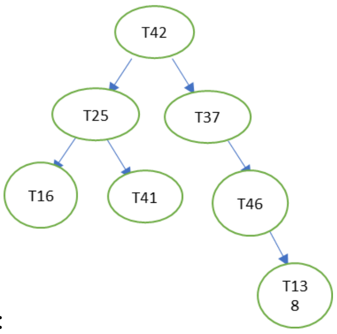
Ts12:



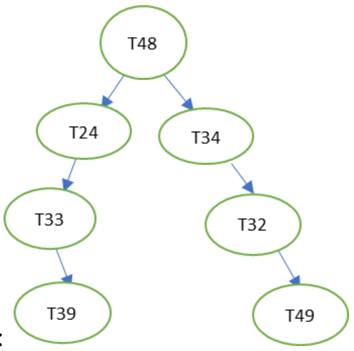
Ts13:



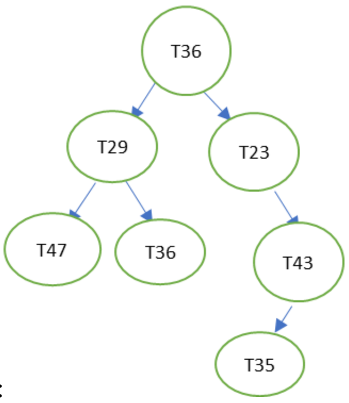
Ts14:



Ts15:

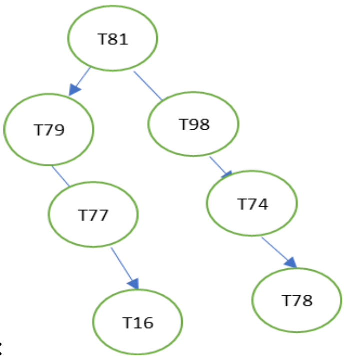


Ts16:

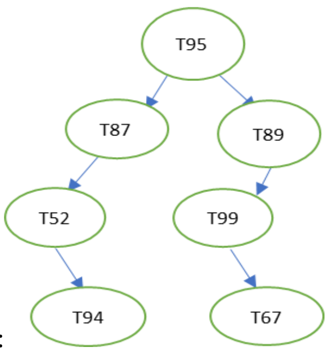


Ts17:

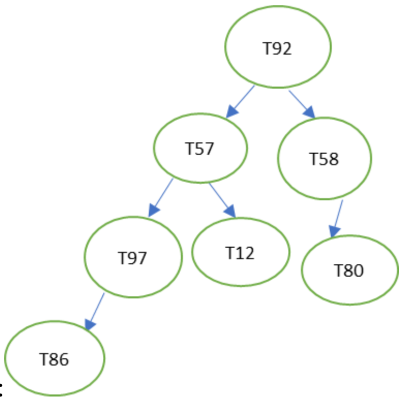
B. Trees constructed using T_{s2} set:



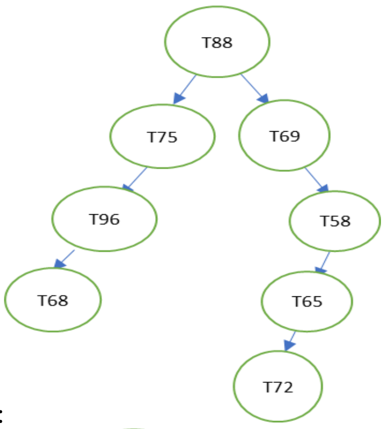
Ts21:



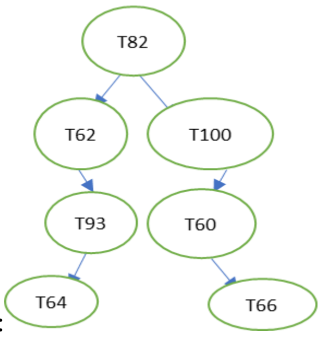
Ts22:



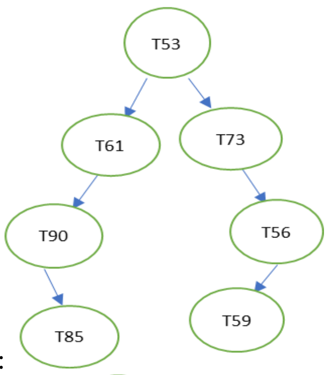
Ts23:



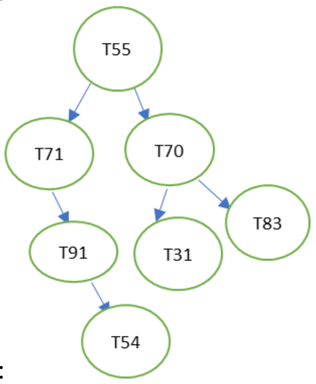
Ts24:



Ts25:



Ts26:



Ts27:

VI. FRIENDSHIP BETWEEN TREES

s.no	Friendship values	Low/High
1	Ts11-Ts21=0.0625	(low)
2	Ts11-Ts22=0.0625	(low)
3	Ts11-Ts23=0.4375	(low)
4	Ts11-Ts24=0	(high)
5	Ts11-Ts25=0.0625	(low)
6	Ts11-Ts2=0.0625	(low)
7	Ts11-Ts27=0.4375	(low)
8	Ts12-Ts21=0	(high)
9	Ts12-Ts22=0	(high)
10	Ts12-Ts23=0.375	(low)
11	Ts12-Ts24=0.0625	(low)
12	Ts12-Ts25=0	(high)
13	Ts12-Ts26=0	(high)
14	Ts12-Ts27=0.375	(low)
15	Ts13-Ts21=0.375	(low)
16	Ts13-Ts22=0.375	(low)
17	Ts13-Ts23=0	(high)
18	Ts13-Ts24=0.4375	(low)
19	Ts13-Ts25=0.375	(low)
20	Ts13-Ts26=0.375	(low)
21	Ts13-Ts27=0	(high)
22	Ts14-Ts21=0.75	(low)
23	Ts14-Ts22=0.75	(low)
24	Ts14-Ts23=0.375	(low)
25	Ts14-Ts24=0.8125	(low)
26	Ts14-Ts25=0.75	(low)
27	Ts14-Ts26=0.75	(low)
28	Ts14-Ts27=0.375	(low)
29	Ts15-Ts21=0.375	(low)
30	Ts15-Ts22=0.375	(low)
31	Ts15-Ts23=0	(high)
32	Ts15-Ts24=0.4375	(low)
33	Ts15-Ts25=0.375	(low)
34	Ts15-Ts26=0.375	(low)
35	Ts15-Ts27=0	(high)
36	Ts16-Ts21=0(high)	(high)
37	Ts16-Ts22=0(high)	(high)
38	Ts16-Ts23=0.375(low)	(low)
39	Ts16-Ts24=0.0625(low)	(low)
40	Ts16-Ts25=0(high)	(high)
41	Ts16-Ts26=0(high)	(high)
42	Ts16-Ts27=0.375(low)	(low)
43	Ts17-Ts21=0.375(low)	(low)
44	Ts17-Ts22=0.375(low)	(low)
45	Ts17-Ts23=0(high)	(high)
46	Ts17-Ts24=0.4375(low)	(low)
47	Ts17-Ts25=0.375(low)	(low)
48	Ts17-Ts26=0.375(low)	(low)
49	Ts17-Ts27=0	(high)

The friendship values for different trees in the set TS1 with the trees in set TS2 is obtained and tabulated above. The friendship is decided based on the conditions like:

If the friendship is between the range 0-0.05 then the two trees are said to have high friendship.

If the friendship is equal to 0.05then the two trees are said to have medium friendship.

If the friendship is between the range 0.05-1.00 then the two trees are said to have low friendship

The trees with high friendship value are:

- Ts11-Ts24
- Ts12-Ts21
- Ts12-Ts22
- Ts12-Ts25
- Ts12-Ts26
- Ts13-Ts23
- Ts17-Ts23
- Ts15-Ts23
- Ts15-Ts27
- Ts16-Ts21
- Ts16-Ts22
- Ts16-Ts25
- Ts16-Ts26
- Ts17-Ts27

We will select some trees, from the above tree pairs ,based on their relationship with other trees

Ts11 and Ts24 trees have high friendship, either Ts11 or Ts24 is selected. There will be no change in number of test cases even if we select Ts11 or Ts24.

Ts12 tree is having high friendship with Ts22, Ts25, Ts26 trees. If we select Ts12 tree then we can avoid executing test cases in other three trees. Otherwise the test cases in Ts22, Ts25, Ts26 trees should be executed. This increases the number of test cases to be executed

Ts13 and Ts23 trees have high friendship, either Ts13 or Ts23 is selected.

Ts15 is having high friendship with Ts23 and Ts27.we select Ts15 tree and we can discard remaining tree Ts27 as Ts23 is already selected in the above tree pair.

Ts16 is having high friendship with Ts22, Ts25 and Ts26. We select Ts16 and we can discard remaining three trees.

Ts17 is having high friendship with Ts23 and Ts27. We select Ts17 and discard remaining trees

From the above points the total trees selected are Ts11, Ts12, Ts23, Ts15, Ts16, Ts17, Ts24

The test cases in these trees are executed.

VII. PRIORITISED AND DISCARDED SIMILAR TEST CASES

Dataset Name	Number of Require- ments	Design Diagram	LOC	Number of Test Cases
News-A: An Online News Portal	24	Yes	450	53
Scientific Calculator	8	Yes	784	39
Sparrow: File Reading Software	21	Yes	875	28
Amghotok: A Platform of Marriage	13	Yes	953	22
Painter: A Canvas for Painting Freely	12	Yes	1056	19
POAS: Program Office Automation Software	18	Yes	4067	62

VIII. EXPERIMENTAL RESULTS:

Table 1 : Dataset Information for Test Case Prioritization

Prioritization Parameters	Values
Range of Prioritization	1 to 10
Implementing language	Java
Numbers of requirements in dataset	Varied from 8 to 24
Numbers of Line of Code in dataset	Varied from 450 to 4067
Numbers of Test Cases in dataset	Varied from 19 to 62

Table 2 :Experimental Setup and Determined Constants Values for Implementing different Prioritization Techniques

To gauge the productivity of experiment prioritization strategy, blame identification rates and experiment execution rate both ought to be considered as measurements. Those parameters represent the execution of prioritization methods in two distinctive purpose of perspectives. Definition and exploratory aftereffects of those two execution investigation stages are given underneath.

1) Phase 1: Level of recognized blames after various size of test execution.

S.no	Prioritised test cases sequence	Discarded similar test cases
1.	T_{S12}- T_{30},T_{50},T_{21}, T_{28},T_{14},T_9,T_6	T_{S21}- T_{81},T_{98},T_{74},T_{78},T_{79},T_{77},T_{76} T_{S22}- T_{95},T_{89}, T_{69}, T_{99}, T_{87},T_{52},T_{94} T_{S25}- T_{82},T_{100},T_{60},T_{66},T_{62},T_{93},T_{64} T_{S26}- T53,T73,T56,T59,T_{61},T_{90},T85
2.	T_{S23}- T_{92},T_{58},T_{80}, T_{51}, T_{97},T_{57},T_{86}	T_{S13}- T_{40},T_{44}, T_4, T_{22},T_{11},T_{27},T_{15} T_{S14}- T_7,T_{45},T_3,T_{20},T_{18},T_{17},T_8
3.	T_{S24}- T_{88},T_{69},T_{58},T_{65}, T_{72},T_{75},T_{96},T_{68}	T_{S11}- T_{12},T_{19},T_{26},T_1,T_{10},T_2,T_{31},T_5
4.	T_{S11}- T_{12},T_{19},T_{26},T_1, T_{10},T_2,T_{31},T_5	T_{S24}-T_{88},T_{69},T_{58},T_{65},T_{72},T_{75},T_{96},T_{68}
5.	T_{S16}- T_{48},T_{34},T_{32},T_{49}, T_{24},T_{33},T_{39}	-
6.	T_{S17}- T_{36},T_{23},T_{43}, T_{35},T_{29},T_{47},T_{36}	-

2) Phase 2: Level of experiment execution to distinguish diverse measure of shortcomings

Test Case Prioritization Techniques

Dataset	Without Any testcase prioritiz ation	RSF [7]	AFC [8]	KMT SP
Amghotok	75.00	58.33	83.33	85.33
Scientific Calculator	75.00	75.00	75.00	76.00
Painter	62.50	75.00	75.00	88.50
News A	32.14	92.86	92.86	94.14
Sparrow	66.67	83.33	83.33	84.81
POAS	80.00	90.00	90.00	100.00
Average	65.22	79.09	83.25	88.13

Table 3 : Percentage of Detected Faults after 75% Test Case Execution

Test Case Prioritization Techniques

Test Case Execution	Without Any testcase prioritization	RSF	AFC	KMT SP

0%	0.00	0.00	0.00	0.00
25%	19.80	22.58	25.63	36.09
50%	41.47	52.76	53.85	64.89
75%	68.00	79.09	83.25	86.47

Table 4 : Average Percentage of Fault Detection for Result Comparison

IX. CONCLUSION

We can observe from the above results that our proposed technique has significant improvement over other techniques employed for the test case prioritization technique. The algorithm we have used has ability of finding out nearly 87% of the faults present in a module under test. The future scope is to include even more test constraints and to see if our technique scales well over complex test cases and we are in the process of developing the concept of friendship between test cases for specific design constraints.

REFERENCES

1. S. Elbaum, A. G. Malishevsky, G. Rothermel, "Test case prioritization: A family of empirical studies", IEEE Trans. Softw. Eng., vol. 28, no. 2, pp. 159-182, Feb. 2002.
2. E. G. Cartaxo, F. G. O. Neto, P. D. L. Machado, "Automated test case selection based on a similarity function", Lecture Notes Informat., vol. 7, pp. 399-404, 2007.
3. T. Y. Chen, M. F. Lau, "Test case selection strategies based on boolean specifications", Softw. Testing Verification Rel., vol. 11, no. 3, pp. 165-180, 2001.
4. S. Yoo, M. Harman, "Pareto efficient multi-objective test case selection", Proc. Int. Symp. Softw. Testing Anal., pp. 140-150, 2007.
5. A. Smith, G. Kapfhammer, "An empirical study of incorporating cost into test suite reduction and prioritization", Proc. Symp. Appl. Comput., pp. 461-467, 2009.
6. M. G. Epitropakis, S. Yoo, M. Harman, E. K. Burke, "Pareto efficient multi-objective regression test suite prioritization", UCL London U.K. Dept. Comput. Sci. Tech. Rep. RN/14/01 , 2014.
7. M. J. Arafeen, H. Do, "Test case prioritization using requirements-based clustering", Proc. IEEE Int. Conf. Software. Testing Verification Validation, pp. 312-321, 2013.