2024 IEEE 4th International Conference on ICT in Business Industry & Government (ICTBIG) | 979-8-3315-1898-1/24/$31.00 ©2024 IEEE | DOI: 10.1109/ICTBIG64922.2024.10911525

A Regression Test Case Prioritization Technique for Web Application Using User Session Data

Kunal Pilley
Faculty of Communication Engineering
Military College of Telecommunication Engineering
Indore, India
kunal.pilley08@gmail.com

Rajib Mall
Department of CSE
Shiv Nadar Institute of Eminence
Greater Noida, India
rmall1961@gmail.com

Sourav Biswas
Centre for IoT
Siksha O Anusandhan University
Bhubaneswar, India
bsws.sourav@gmail.com

Vishal Mamgain
Faculty of Communication Engineering
Military College of Telecommunication Engineering
Indore, India
b21758@mail.com

Rajesh Verma
Faculty of Communication Engineering
Military College of Telecommunication Engineering
Indore, India
rv022024@gmail.com

S K Vishvakarma
Department of EE
IIT Indore
Indore, India
skvishvakarma@iiti.ac.in

***Abstract*—We propose a novel regression test case prioritization technique (RTCP) for web applications exploiting user session data and hamming code distance. The proposed technique utilises user session data to reorder test cases such that test cases which cover the 'most frequently accessed subset of pages together' are given higher priority. The coverage information of user session data along with its frequency appearance and test cases are converted to hamming code. These hamming codes are used to calculate the coverage similarity between user access pattern and the test cases. Further, the hamming code distance is incorporated with frequency of appearance to calculate a weighted average distance among user session data and test cases. Finally, the test cases with a lesser weighted average distance are given higher priority.**

***Index Terms*—Regression Testing, Regression Test Case Prioritization, Web Application, Hamming Code Distance**

I. Introduction

After modifying software or fixing a bug, developers frequently seek confirmation that unmodified code has not been negatively impacted. Regression errors happen when such unaltered code is negatively impacted. Hence, any errors or bugs that may have snuck into the earlier functionalities following the relevant change are referred to as regression issues/bugs, and related validation testing for such issues is termed regression testing.

Regression testing can be done using different approaches such as Retest All , Regression test selection, Test Suite Reduction and Test case prioritisation(TCP). Test case prioritisation technique is reordering of the test cases after assignment of priority based on some selected criterion and accordingly changing the execution order of the test cases from how they were originally run in initial test suite. This technique aims to increase the the possibility that if test cases are prioritised in some manner, they would meet the objective more closely than otherwise.

Several TCP techniques has been proposed by various researchers for tradition software systems. However so far no work for test case prioritisation for web application using user session data and hamming code distance has been produced.

For web applications, we propose a novel TCP approach utilising user session data and hamming code distance. Our technique uses user session data to identify unique subsets of URL which are frequently accessed together along with their frequency of appearance. Information obtained from user session data and test cases are converted to hamming code based on their coverage information. These hamming codes are used to calculate the similarity of coverage information between user access pattern and the test cases. Further the hamming distances is incorporated with frequency of appearance to calculated a weighted average distance among user session data and test cases. Finally the test cases with lesser weighted average distance are given higher priority. The novelty and major contributions of our works can be briefly summarised as following points:

- We have proposed a novel TCP approach specifically for web-applications using user session data and hamming code distance.
- The proposed techniques incorporates the merits of coverage-based and frequency-based method in single approach.

II. Related Works

Researchers, in past, have proposed large number of TCP techniques. These TCP techniques can be generalised into various categories such as Coverage based [1] [2] [3], Model based [4] [5] [6] [4], Fault based [7], Requirement based [8] [8] [9] and Algorithm based techniques [10] [11] [12] .

For general purpose application/software solutions prioritisation based on various criteria has been proposed adequately. The target area of the report is web bases applications with special emphasis given on the user session data. Few important criteria and works proposed earlier for the web based application by researchers are discussed subsequently.

Toofani et al. [13] proposes Bayesian network-based method for prioritising test cases for web-application. Sampath et al. [14] proposed that test cases which cover most frequently accessed sequence of pages should be given priority over the test cases which cover less frequently accessed sequences of pages. They proposed two techniques namely Most Frequently Accessed Sequence (MFAS) and All Accessed Sequences (AAS). Here authors considered sequences in terms of base requests for pages while ignoring any parameter-value pairs. The MFAS approach utilises only the most frequently accessed request sequence and prioritise test cases in decreasing order of appearance of it. The test case containing the most frequently accessed request sequence for highest number of times will be given highest priority for execution. AAS approach considers the frequency of all accessed sequence and prioritise test cases based on all the sequences cumulatively.

In the same work Sampath et al. [14] have proposed a rather simpler way of prioritisation using base request , test cases are prioritised by the no of base request they contains. This method consider the duplicate values as well. The test cases are arranged in ascending **Req-StoL** and descending **Req-LtoS** order of no of base request they contain.The authors propose that arranging test cases based on their total number of base requests can significantly affect the fault detection rate of the ordered test suite, as the number of requests within a test case partially determines the extent of application code exercised by that test case.

Sampath et al. [15] furthered their work on web-application based TCP technique [14][16][17] by combining test case prioritisation approach with test suite reduction to improve effectiveness of reduced test suite in time constrained situations. In the latest work khaleel et al. [18] suggested artificial intelligence based RTCP and Rawat et al. [19] proposed machine learning based RTCP technique. Bakr et al. [20] has provided a Systematic Literature Review of RTCP.

III. Proposed Method

With capturing the user session data and prioritising the test cases based on them, the test suite will provide more relevant feedback to testers by detecting potential faults exposed by current usage .The proposed method hypothesise that :

”The test case which cover ’most frequently accessed subset of pages together’ are given priority for execution above the test cases that exercise the ’less frequently accessed subset of pages together’ with an aim of increased rate of fault detection.”

Use of user session data for testing purposes has been studied and proposed earlier as well.Sampath et al. [21] and Sprenkle et al. [22] have used usage logs to generate user session based test cases. Sampath et al. [14] has also presented frequency based prioritisation which for two out of three applications showed the highest APFD score. However the calculations were for 10% of test-cases and subjected to exact sequence of access of pages, not the subset of pages, and conveniently only limited length sequence were considered.

The Proposed method has following advantage over previous works :

- Incorporating advantages of coverage based method and frequency based method in single approach.
- Inclusion of user perceived reliability in terms of user accessed sequences.
- Not unnecessary strict as previous works which consider exact sequences of access of pages .
- Utilises the sequences obtained by user session data unlike previous works which obtain sequence from the already existing test suites.
- No restriction on length of the sequences unlike previous works which utilises a fixed length for sequences.

The overview to the approach can be be given by a step by step representation as given in figure 1. The action taken in steps and their significance is given in succeeding subsections.

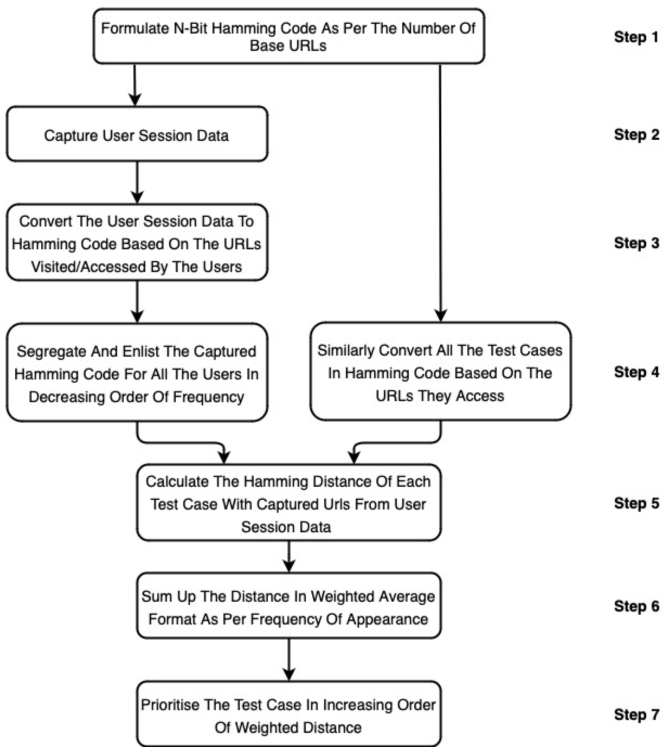
Formulate N-Bit Hamming Code As Per The Number Of Base URLs
Step 1
Capture User Session Data
Step 2
Convert The User Session Data To Hamming Code Based On The URLs Visited/Accessed By The Users
Step 3
Segregate And Enlist The Captured Hamming Code For All The Users In Decreasing Order Of Frequency
Similarly Convert All The Test Cases In Hamming Code Based On The URLs They Access
Step 4
Calculate The Hamming Distance Of Each Test Case With Captured Urls From User Session Data
Step 5
Sum Up The Distance In Weighted Average Format As Per Frequency Of Appearance
Step 6
Prioritise The Test Case In Increasing Order Of Weighted Distance
Step 7

Fig. 1. Flow Chart of Methodology

A. Formulation of n-bit hamming code

firstly an *n-bit* hamming code is formulated as per the number of base URLs/pages in the web application. if a web application has 10 pages in total the corresponding hamming code will have 10 bits. Each URL in the web application is assigned a specific position in the *n-bit* hamming code. Assigning fixed position to each URL enables the use of hamming code as hamming code distance can only be calculated on equal length sequences. It also enables the conversion of user session data and the test cases in *n-bit* hamming code which is further utilised in calculating the similarity between session data and the test cases. This is represented by Figure 1- **Step 1** of the flow diagram.

B. Conversion of User Session Data to Hamming Code Along with Frequency of Appearance

With the access to the user session data, user access sequences can easily be obtained i.e. what-all pages/URLs were accessed/requested by the user in a particular session. In this section all the user access sequences are taken one by one by and they are converted to corresponding hamming code. If a particular page/URL has been visited/requested the corresponding bit in *n-bit* hamming code is set to 1 and bits corresponding to unvisited pages remains zero. Thus the coverage information gets converted to hamming code. Further we segregate the hamming code so found because different session can provide us with same hamming code. This gives us the frequency of the appearance of the each unique hamming code. The frequency gives us the insight of user perspective of usage of the web application. This is represented by Figure 1- **Step 2 - step 4** of the flow diagram and Algorithm 1.

C. Conversion of Test Cases to Hamming Code

In similar manner the test cases are also converted to hamming code. Which presents us with two sets of hamming code one from session data along with frequencies, and second from the test cases. Thus the coverage information has been converted to hamming code and frequency gives us the user perceived importance of the subset of pages. This is represented by Figure 1-**step 4** of the flow diagram and Algorithm 2.

D. Prioritisation Metric and Calculation of Weights

After Conversion of the coverage information into Hamming codes, We calculate the distance between each pair of 'unique user session generated hamming codes' and 'test case generated hamming codes'. n test cases and m unique user session sequence will provide us with $n \times m$ pairs of hamming code distances. Which can be represented as an $n \times m$ distance matrix.

Each unique user session sequence associates a frequency with it. The fraction of that frequency out of total occurrence of sessions will provide us with the weightage of that sequence. It is natural that sum of all the weightages will be equal to 1. Subtracting these weightages from 1 will give us the corresponding multiplication factor of each sequence.

Finally the hamming distance between the test cases and unique user session sequences along with the multiplication factors of sequences will be used to calculate a $WeightedDistance$ to prioritise test cases. The test case with lesser $WeightedDistance$ will be the prioritised over other test cases. The Prioritisation Metric can be given by the following expression :

$$WeightedDistance(T_i) = \sum_{j=1}^{m} MF_{S_j} \times HD_{T_i S_j}$$

T_i refers to i-th test case of the test suite
MF_{S_j} refers to the multiplication factor of j-th user sequence
$HD_{T_i S_j}$ refers to the hamming distance between i-th test case j-th user sequence
i = $1 \ldots n$ where n is total no of test cases
j= $1 \ldots m$ where m is total no of unique sequence

The prioritisation metric is referred as **USDHCD** (User Session Data - Hamming Code Distance). With the calculation of the weighted distance the prioritisation is completed. The test cases can be prioritised in ascending order of weighted distance with minimum weighted distance given the highest priority. This is represented by Figure 1, **step 5 - Step 7** of the flow diagram and Algorithm 3.

IV. Algorithmic Representation of Methodology

Algorithm 1 refers to the Figure 1, **Step 1- step 4** of the flow diagram, depicting user sequence conversion to hamming code and calculation of associated frequencies. In step 2-5 of the Algorithm 1, it takes all user accessed sequences U, set of URLs/Pages in web-application URL and hamming code length n (total number of URLs in web application) as the inputs. As output, It provides unique hamming code representation of User accessed sequence S, their corresponding frequencies of appearance F with multiplication factors MF_S. In Steps 11-18, each user accessed sequence is converted into the corresponding hamming code representation. In Steps 19-25, the converted hamming code is checked for its frequency. If the produced hamming code is already present in S its frequency is increased by 1, otherwise a new entry to the S is made and corresponding frequency is set to 1. In Steps 26-28 multiplication factors MF_S of each unique hamming code sequence in S, is calculated.

Algorithm 2 refers to the Figure 1, **Step 4** of the flow diagram, depicting conversion of test cases to hamming code. In step 2-5 of the Algorithm 2, it takes test suite TS, set of URLs/Pages in web-application URL and hamming code length n (total number of URLs in web application) as the inputs. As output, It provides hamming code representation of each test case as T. In Steps 9-17, each test case is processed one by one and converted to corresponding hamming code representation based on URLs it contains/accesses.

Algorithm 3 refers to the Figure 1, **Step5 - Step7** of the flow diagram, depicting calculation of $WeightedDistance$ of test cases and assignment of final priorities. In step 2-5 of the Algorithm 3, it takes test suite TS, unique hamming code representation of user accessed sequence S, their corresponding multiplication factors MF_S and hamming code representation of test case T as inputs. Where S and MF_S are outputs of Algorithm 1 and T is output of Algorithm 2. As output, It provides hamming distance Between *i*-th test case and *j*-th unique user sequence $HD_{T_iS_j}$ and finally the Prioritised Test Suit PT. In Steps 9-17, $HD_{T_iS_j}$ is calculated. Which is a simple XOR operation between *i*-th test case and *j*-th unique user sequence and $CountOnes$ counts the number of ones present into the resultant which is taken as hamming distance. In Steps 18-20, the $HD_{T_iS_j}$ and multiplication factor MF_S is used to calculate the $WeightedDistance$ associated with each test case. Step 21-22, prioritise the test cases in increasing order of $WeightedDistance(T_i)$ and produce the final output as PT.

Algorithm 1 Conversion Of User Access Sequence To Hamming Code

```
 1: Input Parameters :
 2: $U$ : All User Access Sequence form Individual Sessions
 3: $URL$: Set of URLs/Pages in Web-application
 4: $n$ : Hamming Code Length, No of URL in web application
 5: $HU_i$ : Hamming Code Representation of i-th User Access Sequence - $U_i$
 6: Output :
 7: $S$ : Unique Hamming Code Representation of User Accessed Sequences
 8: $F$ : Corresponding Frequency of Unique Hamming Code Representation of User Accessed Sequences
 9: $MF_S$ : Multiplication Factor of Unique Hamming Code Sequence
10: Computation :
11: Assign Each URL a Unique Position in n-Bit hamming code
12: for all $U_i \in U$ do
13:     Set Each Bit of $HU_i \leftarrow 0$            ▷ Initialise each bit
14:     for all $URL_j \in URL$ do
15:         if $URL_j\ PresentIn\ U_i$ then
16:             $HU_i[j] \leftarrow 1$
17:         end if
18:     end for
19:     if $HU_i \notin S$ then
20:         Add a New Entry for Unique Hamming Code in $S$
21:         Set Corresponding frequency to 1
22:     else
23:         Increase Corresponding Frequency of Existing Unique Hamming Code by 1
24:     end if
25: end for
26: for all $S_i \in S$ do
27:     $MF_{S_i} \leftarrow 1 - (\frac{F_i}{\sum F_i})$
28: end for
```

Algorithm 2 Conversion Of Test Cases To Hamming Code

```
 1: Input Parameters :
 2: $TS$ : Test Suite
 3: $URL$ : Set of URLs/Pages in Web-application
 4: $n$ : Hamming Code Length, No of URL in web application
 5: $T_i$ : Hamming Code Representation of i-th Test Case $T_i$
 6: Output
 7: $T$ : Hamming Code Representation of Test Cases
 8: Computation :
 9: Assign Each URL a Unique Position in n-Bit hamming code
10: for all $TS_i \in TS$ do
11:     Set Each Bit of $T_i \leftarrow 0$            ▷ each bit
12:     for all $URL_j \in URL$ do
13:         if $URL_j\ PresentIn\ T_i$ then
14:             $T_i[j] \leftarrow 1$
15:         end if
16:     end for
17: end for
```

Algorithm 3 Prioritisation Of Test Cases Using Hamming Code Distance And Weighted Average

```
 1: Input Parameters :
 2: $TS$ : Original Test Suit
 3: $S$ : Unique Hamming Code Representation of User Accessed Sequence            ▷ Using Algo 1
 4: $T$ : Hamming Code Representation of Test Cases ▷ Using Algo 2
 5: $MF_S$ : Multiplication Factor of Unique Hamming Code Sequence            ▷ Using Algo 1
 6: Output :
 7: $HD_{T_iS_j}$ : Hamming Distance Between i-th Test Case and j-th Unique User Sequence
 8: $PT$ : Prioritised Test Suit
 9: Computation :
10: Capture User Session Data
11: Convert User Session Data To Unique Hamming Codes With Frequency            ▷ Using Algo 1
12: Convert Test Cases To Hamming Code            ▷ Using Algo 2
13: for all $T_i \in T$ do
14:     for all $S_j \in S$ do
15:         $HD_{T_iS_j} \leftarrow CountOnes(T_i \oplus S_j)$
16:     end for
17: end for
18: for all $T_i \in T$ do
19:     $WeightedDistance(T_i) \leftarrow \sum_{j=1}^{m} MF_{S_j} \times HD_{T_iS_j}$
20: end for
21: Prioritise The Test Case In Increasing Order Of $WeightedDistance(T_i)$
22: return ($PT$)
```

V. Experimentation and Results

TCP Performance Evaluation Metric : To access the performance of test case prioritisation technique for its effectiveness the APFD metric is used [2]. APFD is given as follows:

$$APFD = 1 - \frac{\sum_{j=1}^{m} TF_j}{nm} + \frac{1}{2n}$$

TF_j refers to 1st test case of prioritised test suite which detect j-th fault
n is total no of test cases
m is total no of fault detected by test suite

A. *Data Set*

The experimentation has been conduct on 5 student developed web applications namely **GetAlrt** - a price tracking web application which keeps an account of varying price across multiple e-commerce website for an item and provides statical details in change of price and alert the user if the price comes down below a specified value , **BookLine** - an e-commerce web application specifically developed for an online Book store for searching and buying books , **IMS-BookLine** - an inventory management system for physical book stores to facilitate the storage, stock taking and order management and audits, **CollegeStuff** - a web application for students to cater for academic and non-academic needs such as books, laptops, smartphones, hostels, podcasts and scholarship recommendations etc. and lastly **Refinne** - an app for searching restaurants and cafes along with availing discount coupons and deals on dining-in, take-aways, making reservations and placing orders. The Websites were selected with an aim to keep the experiment diversified with different forms of web applications. The requirement analysis contributed to creation of test cases. Additional test cases were also added based on user access sequence. The fault matrix was populated against the test cases with test case execution and successive user interactions which revealed the faults. Table I displays the no of pages, sessions, test cases and faults for each web-application under consideration.

TABLE I
DATA-SET FOR EXPERIMENTAL EVALUATION

Data-Set				
Student Developed Web Application	**Pages**	**Sessions**	**Test Cases**	**Faults**
GetAlrt	8	2862	350	400
BookLine	15	2240	275	350
IMS-BookLine	20	1870	220	330
CollegeStuff	20	1610	200	300
Refinne	15	1400	200	225

B. *Experimental Results*

We contrast the outcomes of USD-HCD with those of four methods: No-Prioritization (NP), Random Prioritization(RP), Req-LtoS [14] , and All Accessed Sequence (AAS) [14] of resgression test case prioritisation. No-prioritization technique considers the original order of test suite and does not change the ordering for execution. Random prioritization technique randomises the order of test cases to achieve better results. Req-StoL imployes prioritization methodology in which test cases are prioritised according to the total number HTTP requests they contain. This may consider both i.e. Descending Order of Length - Req-LtoS and Ascending Order of Length - Req-StoL [14].

The USD-HCD incorporates advantages of coverage based method and frequency based method in single approach. Our proposed technique USD-HCD enjoys similarity with both Req-StoL/Req-LtoS and AAS, as it consider the accessed pages/URLs similarly to former and employs the frequency of requested sequence similarly to the latter. However USD-HCD is not unnecessary strict as AAS which consider exact sequences of access of pages whereas it utilises subset of request as whole. Also unlike AAS, which obtains request sequences from the already existing test suites , USD-HCD uses user session data to include the notion of client perceived reliability. USD-HCD offers no restriction on length of the sequences whereas in the research work AAS as taken the sequence length as two URLs.

TABLE II
APFD SCORES

APFD					
Web Application	**NP**	**Random**	**Req-LtoS**	**AAS**	**USD-HCD**
GetAlrt	0.583	0.643	0.593	0.649	0.814
BookLine	0.461	0.483	0.71	0.791	0.951
IMS-BookLine	0.398	0.51	0.81	0.842	0.937
CollegeStuff	0.573	0.41	0.79	0.815	0.913
Refinne	0.491	0.573	0.625	0.751	0.891
Average	0.492	0.523	0.705	0.769	0.901

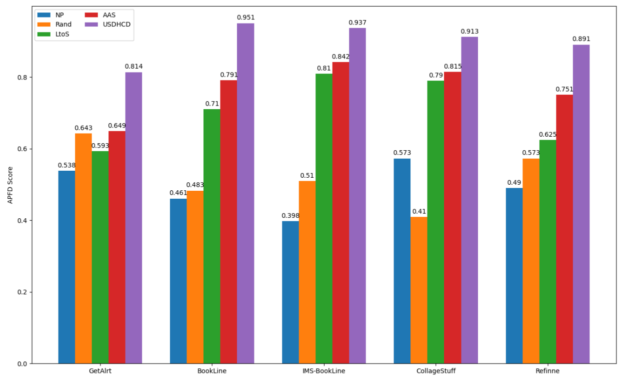

Fig. 2. APFD Scores - Grouped Bar Plot

Table II shows the obtain APFD scores for all the website considering different prioritisation technique figure 2 represents the grouped bar plot diagram of the same . Table II demonstrates that USD-HCD TCP method achieved the highest APFD value 0.951 for BookLine and the lowest APFD value 0.814 for GetAlrt. For almost similar parameter of no of pages, session, test cases and faults USD-HCD gives a higher APFD values for IMS-BookLine over CollegeStuff. It can be observed from data-set table I, for same no of pages IMS-BookLine has slightly higher values of sessions, test cases and faults over CollegeStuff that might have led USD-HCD to achieved highest APFD for it. APFD values for BookLine and Refinne suggest that, When there are more numbers of sessions to be considered , USD-HCD demonstrates a notable enhancement in early fault detection compared to other methods for similar no of test cases and faults. It should also be noted that for Bookline, USD-HCD provide the maximum APFD score across all web applications and TCP techniques in consideration, incidentally Bookline happens to have highest no of user session under consideration.

However, despite having good number of sessions under consideration for GetAlrt, USD-HCD shows significant drop in APFD score and achieves the lowest APFD score over rest of the web applications. Similarly ReqLtoS and AAS also showes significant drop in achieved APFD score for GetAlrt over the rest of the application and achieve the lowest APFD score. This may be contributed by the fact that GetAlrt has the lowest number of pages among all the web-applications which makes the prioritisation criteria i.e. number of URLs in test cases in Req-StoL, All Accessed Sequences in test cases for AAS and User session data for USD-HCD, less distinct. Which leads to assigning similar priority to multiple test cases and resembling the behavior of Random Prioritisation providing low APFD results . It is also evident that Random outperforms Req-StoL and provide similar APFD value for AAS for GetAlrt.

The Random prioritization achieved a higher APFD than NP across all web applications, with the exception of CollageStuff. Additionally, it can be noted that Req-LtoS, AAS, and USD-HCD outperform both NP and Random in all cases, except for one instance where Random surpasses Req-StoL for GetAlrt. However, our USD-HCD technique consistently outperforms all other methods across the web applications. As shown in Table II, USD-HCD achieves the highest average APFD score (0.901), while NP records the lowest average APFD (0.492). Table II shows that USD-HCD showed highest average APFD (0.901) and NP showed lowest Average APFD score (0.492). The average APFD scores in decreasing order are : $USD{-}HCD(0.901) > AAS(0.769) > Req - LtoS(0.705) > Random(0.523) > NP(0.492)$. A graphical representation of the same is given by figure 3.

Figure 4 illustrates the APFD scores obtained by the prioritization methods under evaluation. From Figure 4, it can be observed that both the minimum and maximum values for NP and Random are significantly lower than the corresponding values for the other three techniques. , also their maximum APFD score is significantly lower than the lowest APFD score of AAS and USD-HCD. USD-HCD attains the highest median among all the techniques, indicating that it generally outperforms the other methods under consideration. Additionally, USD-HCD has the smallest interquartile range (IQR) of the five techniques, signifying that it demonstrated the most consistent performance across all the web applications evaluated. AAS also has short IQR implying its consistency and efficacy. It is also observed that both AAS and USD-HCD has outlier values in the box plot. These outliers are contributed by the fact that both the approaches have performed significantly low for GetAlrt , that has lowest number of pages among all the web-applications which made the prioritisation criteria less distinct, and they achieved their minimum APFD scores for GetAlrt. Req-LtoS has the highest IQR indicating its inconsistent performance over different web-Applications however its median APFD value is still higher that the maximum values of both NP and Random.

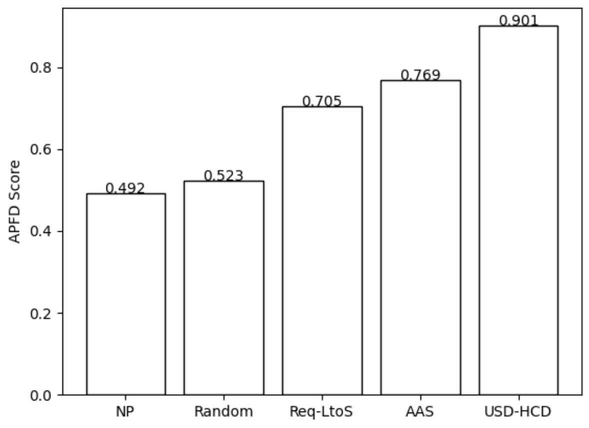

Fig. 3. Average APFD Scores

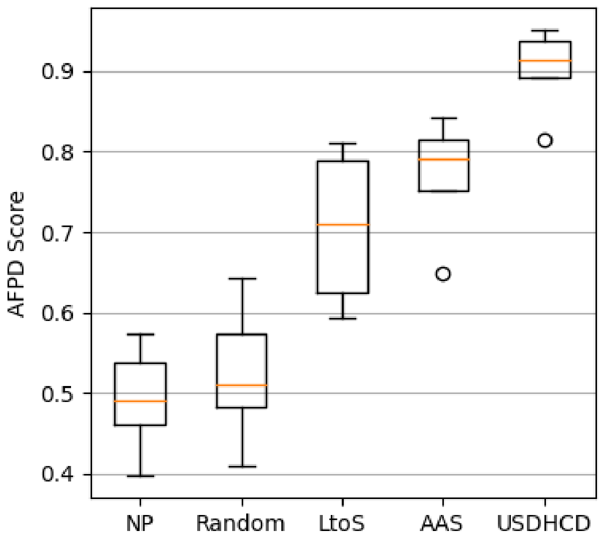

Fig. 4. APFD Scores - Bar Plot

Table III shows the average relative improvement achieved by the USD-HCD over other methos under consideration for all the web-applications in consideration . For all of the projects being tested, we can see that USD-HCD exhibits a sizable relative improvement over Random and NP prioritising. For NP and Random prioritisation, USD-HCD has achieved highest relative improvement as 135.42% for IMS-BookLine and 122.68% for CollegeStuff respectively and

TABLE III
RELATIVE IMPROVEMENT IN APFD SCORES

Average Relative Improvment					
Web Application	**NP(%)**	**Random(%)**	**Req-LtoS(%)**	**AAS(%)**	**Average(%)**
GetAlrt	51.30	26.59	37.26	25.42	35.14
BookLine	106.29	96.89	33.94	20.22	64.33
IMS-BookLine	135.42	83.72	15.67	11.28	61.52
CollegeStuff	59.33	122.68	15.56	12.02	52.40
Refinne	81.83	55.49	42.56	18.64	49.63
Average(%)	86.83	77.07	29.00	17.52	-

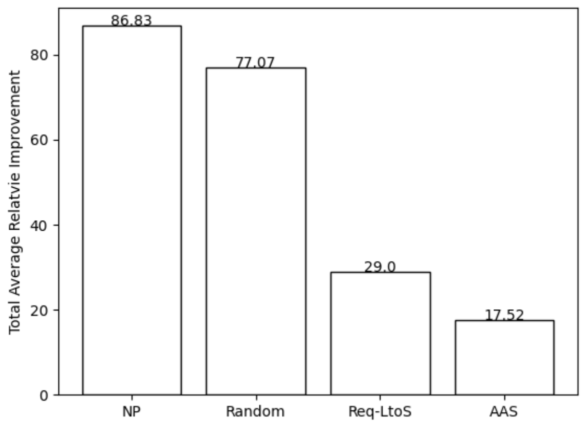

Fig. 5. Average Relative Improvement

lowest relative improvement as 51.30% and and 26.59% for GetAlrt, whereas lowest relative improvements are also very significant in values. Overall USD-HCD has achieved highest relative improvement (135.42%) over NP for IMS-BookLine and lowest relative improvement (11.28%) over AAS also for IMS-BookLine. The decreasing order of average relative improvement is : $NP(86.83\%) > Random(77.07\%) > Req - LtoS(29.00\%) > AAS(17.52\%)$. A graphical representation of the same is given by figure 5.

If we consider the average relative improvement for each web-application, BookLine shows the highest average relative improvement of 64.33% , where as GetAlrt shows the lowest average relative improvement of 35.14% contributed by the fact that both NP and random has given comparatively good scores of APFD value and USD-HCD has given its lowest APFD value for GetAlrt. IMS-Bookline has also shown a score of 61.53 %, a value very close to highest average relative improvement per web-application basis. Barring GetAlrt which has least no of pages, rest of the web-applications has shown to achieve average relative improvement pursuant to the session count.

C. Threats to Validity

Like any research, this study has certain limitations. This section outlines the key limitations that may have influenced the findings ands future course of action to address the same. However some of these limitations are unavoidable consequences of experimental study being in the early stage and the decision to use controlled experimentation. Given that this is the first study that quantifies fault detection effectiveness of the proposed approach, following limitations may be acceptable with resolve of author to address them in the future :

- In this study, we only use five student developed applications for experimentation , which we chose based on their availability. It is necessary to test this empirical comparison using more substantial, professionally hosted, real-world online apps other than student developed applications. A more broadly generalised experiment could include a number of programs of varying domains. The experiment's external validity is threatened by the fact that the representation of applications and data collection are internally developed and in-house in nature.

- The faults and test cases were generated and structured in a similar fashion, which may not fully represent the actual faults or other professional hosted applications, since naturally occurring faults are preferred.

VI. FUTURE WORK

Despite threats to validation authors did employ reasonable constructs for experimentation that have been used before in a number of similar and related empirical contexts. To reduce the threat to effectiveness, inspection of the elements and numerous runs were carried out. Also the fault matrix creation approach used, were equivalent to those that were often used in past studies and are accepted research standard. The threats to validity will be addressed in future work by choosing more appropriate data set from professionally hosted and tested, real-world online applications. Multiple web-applications from various domains and sizes with different target users would be used in experiment to make it more generalised. Currently the result is compared either with No-Prioritisation and Random prioritisation or with approaches exclusive to the web application, in future it may be compared against other popular prioritisation techniques as given in the related works section.

VII. CONCLUSION

This paper focuses it approach on the availability of user session data and its utilisation. The technique differs from the existing techniques as to it utilises the user session data and use hamming code to leverage both the coverage and frequency of access. The proposed technique introduces a metric called $WeightedDistance$ and prioritisation is denoted by USD-HCD. We evaluated the efficacy of our suggested strategy to existing prioritisation techniques including techniques specific to web-applications . The experimental results shows 86.83% average relative improvement in APFD score over NP, 77.07% over Random prioritisation, 29.0% over Req-LtoS and 17.52% over AAS. The results suggest effectiveness of the approach. However there are threats to the validity which gives us the scope of future work.

References

[1] G. Rothermel et al. "Prioritizing test cases for regression testing". In: *IEEE Transactions on Software Engineering* 27.10 (Oct. 2001), pp. 929–948. ISSN: 0098-5589. DOI: 10.1109/32.962562.

[2] G. Rothermel et al. "Test case prioritization: an empirical study". In: *Proceedings IEEE International Conference on Software Maintenance - 1999 (ICSM'99). 'Software Maintenance for Business Change' (Cat. No.99CB36360)*. 1999, pp. 179–188. DOI: 10.1109/ICSM.1999.792604.

[3] Árpád Beszédes et al. "Code coverage-based regression test selection and prioritization in WebKit". In: *2012 28th IEEE International Conference on Software Maintenance (ICSM)*. 2012, pp. 46–55. DOI: 10.1109/ICSM.2012.6405252.

[4] Chhabi Rani Panigrahi and Rajib Mall. "Model-Based Regression Test Case Prioritization". In: *Information Systems, Technology and Management*. Ed. by Sushil K. Prasad et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 380–385. ISBN: 978-3-642-12035-0.

[5] Chhabi Rani Panigrahi and Rajib Mall. "An approach to prioritize the regression test cases of object-oriented programs". English. In: *CSI Transactions on ICT* 1.2 (2013), pp. 159–173. DOI: 10.1007/s40012-013-0011-7.

[6] Chhabi Rani Panigrahi and Rajib Mall. "A heuristic-based regression test case prioritization approach for object-oriented programs". In: *Innovations in Systems and Software Engineering* 10 (2014), pp. 155–163.

[7] Sourav Biswas et al. "A regression test case prioritization technique targeting 'hard to detect' faults". In: *International Journal of System Assurance Engineering and Management* 13.3 (June 2022), pp. 1066–1081. DOI: 10.1007/s13198-021-01385-. URL: https://ideas.repec.org/a/spr/ijsaem/v13y2022i3d10.1007_s13198-021-01385-4.html.

[8] Md. Junaid Arafeen and Hyunsook Do. "Test Case Prioritization Using Requirements-Based Clustering". In: *2013 IEEE Sixth International Conference on Software Testing, Verification and Validation*. 2013, pp. 312–321. DOI: 10.1109/ICST.2013.12.

[9] H. Srikanth, L. Williams, and J. Osborne. "System test case prioritization of new and regression test cases". In: *2005 International Symposium on Empirical Software Engineering, 2005*. 2005, 10 pp.-. DOI: 10.1109/ISESE.2005.1541815.

[10] Ripon K. Saha et al. "An Information Retrieval Approach for Regression Test Prioritization Based on Program Changes". In: *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*. Vol. 1. 2015, pp. 268–279. DOI: 10.1109/ICSE.2015.47.

[11] R. Lachmann. "12.4 - Machine Learning-Driven Test Case Prioritization Approaches for Black-Box Software Testing". In: Jan. 2018, pp. 300–309. DOI: 10.5162/ettc2018/12.4.

[12] I. H. Witten et al. *Data mining : practical machine learning tools and techniques*. Amsterdam; London: Morgan Kaufmann, 2017. ISBN: 9780128042915 0128042915.

[13] Abhishek Toofani et al. "Test case prioritisation during web application testing". In: *International Journal of Computer Applications in Technology* 56 (Jan. 2017), p. 230. DOI: 10.1504/IJCAT.2017.10008708.

[14] Sreedevi Sampath et al. "Prioritizing User-Session-Based Test Cases for Web Applications Testing". In: *2008 1st International Conference on Software Testing, Verification, and Validation*. 2008, pp. 141–150. DOI: 10.1109/ICST.2008.42.

[15] Sreedevi Sampath and Renée C. Bryce. "Improving the effectiveness of test suite reduction for user-session-based testing of web applications". In: *Information and Software Technology* 54.7 (2012), pp. 724–738. ISSN: 0950-5849. DOI: https://doi.org/10.1016/j.infsof.2012.01.007. URL: https://www.sciencedirect.com/science/article/pii/S0950584912000183.

[16] Renee C. Bryce, Sreedevi Sampath, and Atif M. Memon. "Developing a Single Model and Test Prioritization Strategies for Event-Driven Software". In: *IEEE Transactions on Software Engineering* 37.1 (2011), pp. 48–64. DOI: 10.1109/TSE.2010.12.

[17] Renée C. Bryce and Atif M. Memon. "Test Suite Prioritization by Interaction Coverage". In: *Workshop on Domain Specific Approaches to Software Test Automation: In Conjunction with the 6th ESEC/FSE Joint Meeting*. DOSTA '07. Dubrovnik, Croatia: Association for Computing Machinery, 2007, pp. 1–7. ISBN: 9781595937261. DOI: 10.1145/1294921.1294922. URL: https://doi.org/10.1145/1294921.1294922.

[18] Shahbaa Khaleel and Raghda Anan. "A review paper: optimal test cases for regression testing using artificial intelligent techniques". In: 13 (Apr. 2023), p. 1803.

[19] Neelam Rawat, Dr Somani, and Arun Tripathi. "Machine Learning Approach for Regression Testing: A Case Study in Markov Chain Model". In: 12 (Mar. 2024), pp. 945–952.

[20] Bakr Ba-quttayyan, Haslina Haslina, and Fauziah Baharom. "Regression Testing Systematic Literature Review – A Preliminary Analysis". In: *International Journal of Engineering and Technology* Vol 7, No 4.19 (2018) (Nov. 2018), pp. 418–424. DOI: 10.14419/ijet.v7i4.19.23176.

[21] Sreedevi Sampath et al. "Applying Concept Analysis to User-Session-Based Testing of Web Applications". In: *IEEE Transactions on Software Engineering* 33.10 (2007), pp. 643–658. DOI: 10.1109/TSE.2007.70723.

[22] Sara Sprenkle et al. "Automated replay and failure detection for web applications". In: Jan. 2005, pp. 253–262. DOI: 10.1145/1101908.1101947.