Journal of Information Systems Engineering and Management
2025, 10(30s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

Identification of Failure Inducing Combinations using Greedy Heuristic Algorithm

Mahipal Chakravarthy G[1], Keerthana V[2], Abhav Garg3, Rekha Jayaram[4], Krishnan Rangarajan[5]
[1]*Department of Computer Science and Engineering, Dayananda Sagar College of Engineering, Bangalore, India*
[2,3,4,5] *Visvesveraya Technological University, Belagavi – 590018*
rekhajayaram20@gmail.com

ARTICLE INFO

Received: 26 Dec 2024

Revised: 14 Feb 2025

Accepted: 22 Feb 2025

ABSTRACT

The successful eminence of Combinatorial Testing can be realized with effective techniques for Fault Localization (FL). This paper presents an effective FL technique that can accurately determine the Failure Inducing Combinations (FICs) using a greedy heuristic approach. The importance of this technique comes from the fact that when an executed test suite is given as input, it localizes the FICs by catering to all-t-way combinations while generating Test Cases based on their probability of passing. The initial study and experimentation are found to be promising, providing scope for rigorous empirical study.

Keywords: *Failure inducing combinations, Combinatorial testing*

1 INTRODUCTION

Combinatorial Testing (CT) is an organized method of software testing where each test case (TCs) consists of parameters combined based on some combination strategy. It is used to detect software failures caused by interaction among various input parameters. CT generated TCs will have combinations of parameters [17]. An interaction TC generation is categorized into combinatorial TC generation, covering arrays TC generation, t-way TC generation and handling of constraints in CT [14]. Parameter combination is considered as an FIC, when all TCs that have this parameter combination fails [18].

For instance, an interaction leading to failure is given in the following code snippet:

```
if (temperature < 20)
{
   //normal code
if (volume > 500)
{ //code that triggers fault }
else{//correct code }
}
else {//normal code}
```

Note that failure is triggered in the snippet given only when both the parameters temperature < 20 and volume > 500 are true.

When such a failure is detected, the combination of parameters that have caused the said failure should be identified and fixed. This is called FL. There exists multiple tools that are already in place that help us with FL like BEN [13] and MIXTGTE [15]. CT based approaches to FL can be classified into adaptive approaches where the TCs will be incrementally updated and non-adaptive approaches where TCs will be pre-computed [12],[16].

Combinatorial TCs generation strategies use algorithms designed based on techniques such as Greedy algorithm, heuristic search, algebraic or mathematical and random techniques [19].

The proposed approach differs from the above mentioned methods in the following ways:

- BEN generates only specific t-way combinations but the proposed approach generates all possible t-way combinations.
- MIXTGTE generates all the possible extra TCs in pinning down a FIC, but the proposed approach generates more TCs - that depend on their probability of passing for each FIC.

True FIC's are those that always generate failures when the said parameters are being introduced. The current paper works towards identifying true FIC's and also generating the probability of a non-true FIC producing failures.

For this to work, we initially take an input set of TCs that are already executed and consist of parameters and pass / fail value. Once the input set is read, the set of passed TCs and failed TCs are separated from each other. All the combinations in the TCs that have passed are tagged as non-FIC's whereas the combinations in the failed TCs are to be worked upon. Likely FIC's are found for the failed TCs and additional TCs are generated by considering the likely FIC's as a subset for every test case. Generate additional TCs which have the least probability of failure compared to all the possible TCs with the current combination. Once the TCs are generated, the pass/fail value of the TCs is observed and if there is any pass value, the combination is not a FIC.

Otherwise, the combination is termed as a "Failure-Inducing Combination" (F-IC) and a new test is run with a different subset until no further tests can be run. At the end of the iterative process, if there still remains a combination that is generating failures, that combination is a true-FIC. This means that every time a particular combination of parameters is used, the result of the TC will be 100% fail.

2 RELATED WORK

Faulty Interaction Characteristic (FIC) and FIC_BS [1] are two techniques that try to discover all the parameter combinations that are present in a failed test. These techniques start by considering one failed test from a CT set. It then uses a systematic approach to generate a reduced number of TCs. These TCs are then executed to identify the F-ICs in the TC that has failed. New TCs are then generated by changing a single value of the failed TC to another valid value for the parameter. When the newly formed TC passes, the initial value of the parameter is a part of the combination that causes a failure because removing this value will make the TC pass. FIC technique generates a new k-number of tests such that for each FIC, "k" is the number of total parameters in the TC.

The version of FIC which uses a binary search technique is FIC_BS. While generating a new TC, FIC_BS modifies the parameter values of k/2 parameters from the failed TC. If the new TC that is generated passes, it looks for combinations in the modified values of k/2. The process is repeated until all the F-ICs of parameters are identified. FIC and FIC_BS work on the assumption that - a changed parameter value will not introduce any new faults.

Two new approaches RI and SRI were introduced by Li et al. [2], which are used to identify combinations that induce a failure by using a method called delta debugging [3]. This method works in a repetitive manner. The first approach starts with one failing TC from the original test suite and uses a FIC_BS like approach to produce a limited number of TCs. The SRI approach takes one failing TC, f as input. It then tries to create a passing TC that is similar to f. SRI uses the knowledge that - the F-IC appeared in the failed TC f, but was not present in the similar passing TC. Hence, its main focus is on the parameters, which are not the same in the passed and failed TCs. SRI could identify F-ICs by producing fewer tests than RI.

[4], [5] gives AIFL and InterAIFL approaches. Here, a set A is formed which contains the suspicious combinations that are first identified as candidates for being inducing. It then comes up with a group of TCs for every failing TC using a strategy called SOFOT [6]. SOFOT works as follows:

- Let k = parameters count.
- For each TC f, generate k TCs by modifying the value of only one participating parameter at that point of given time.
- In comparison with the original TC "f", the new TCs differ in exactly one value - which is randomly chosen from the related domain of parameters.

- Once the newly generated TCs are executed, the parameter combinations that were present in the passing TC are deleted from the original suspicious set A.

Yilmaz et al. [7] gives a ML approach for finding failure-inducing combinations. This approach examines the CT set and then tests their execution statuses. It then constructs a classification tree which is used to identify FICs. An improved version based on this work can be seen in Shakya et al. in [8].

Ghandehari et al. came up with an approach - BEN [9] - which generates statements with ranks depending on their possibility of being faulty by using the result of CT.

3 APPROACH

The current section presents the approach in detail as shown in Fig. 1. We explicate on each step of this approach with help of a synthetic example test suite mentioned in table 1.

Test No.	P-A	P-B	P-C	P-D	TC Status
TC-1	1	2	1	0	Pass
TC-2	1	1	0	1	Pass
TC-3	2	3	1	0	Pass
TC-4	1	3	1	0	Fail

Table 1: Example executed test suite

3.1 Step 1 – Inputs:

The algorithm considers the executed test suite as the input. This test suite is a group of TCs by a specific combination of values given to the inputs.

Consider the SUT has a set “P” with “k” input parameters, given by

P = {p1, p2 , … , pk }.

Let d_i = domain of each parameter pi (d_i = all possible values of pi)

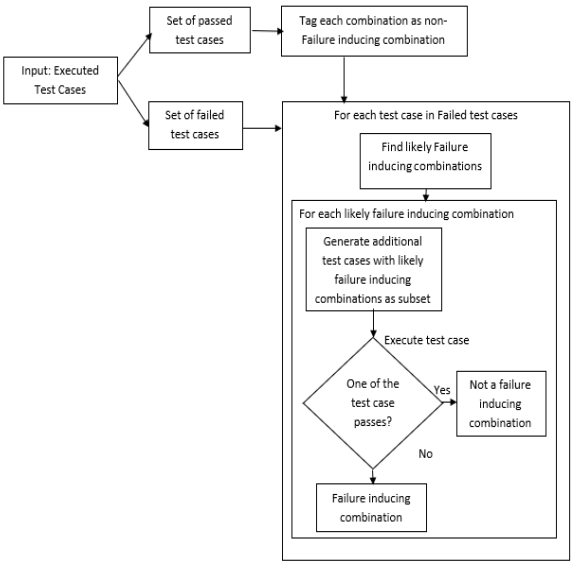

Fig 1: Approach to determine FIC

Let all t-way parameter combinations in P be P_c = {(p1), (p2), …., (p1, p2), ….., (pi, pk), ……, (p1, p2, …., pk)} i.e. for a set of k parameters, there will be (2^k-1) combinations.

Consider the Synthetic TC from table 1 as an example.

From table 1, the parameters set is:

P = {a, b, c, d} and

d_a = {1, 2} d_b = {1, 2, 3} d_c = {0, 1} d_d = {0, 1}

3.2 Step 2 – Segregation and tagging of passed and failed TCs:

In this step we identify combinations in TCs as FIC, n-FIC, LFIC as follows:

The input test set is then categorized into Passed-TCs-set and Failed-TCs-set.

Let T= {v1, v2, …, vk} be a TC containing k input values. Accordingly, V_c contains all values combinations in T.

Let 'S' be a list containing {(P_{ci} ,V_{ci}) ->M) where P_{ci} is the parameter combination, V_{ci} is the value combination for the corresponding parameter in P_{ci} and M is a tag which can take one of three values 'Failure Inducing' , 'Non Failure Inducing' or 'Likely Failure Inducing'.

In the set of passed TCs, there will be no FICs. Therefore, all the t-way combinations in this set are tagged as non-FIC (n-FIC). That is, for each element in V_c, add each combination (P_{ci},V_{ci}) as n-FIC to list S.

In the set of failed TCs, all the t-way combinations of each TC that are not in list S are tagged as Likely FIC(LFIC) and added to the suspicious list of that TC.

As a result, the suspicious list will now comprise of the following kinds of combinations:

1. The combination is failure inducing.
2. The combination consists of a FIC as its subset.
3. The combination is not failure inducing but was not tagged because there was no passed TC with the current combination

For the running example,

The result of test case 1 is 'pass' hence all the t-way combinations {(a = 1), (b = 2), (c = 1), (d = 0), (a = 1, b = 2), (a = 1, c = 1), (a = 1, d = 0), (b = 2, c = 1), (b = 1, d = 0), (c = 1, d = 0), (a = 1, b = 2, c = 1), (a = 1, b = 2, d = 0), (a = 1, c = 1, d = 0), (b = 2, c = 1, d = 0), (a = 1, b = 2, c = 1, d = 0)} are tagged as n-FIC's and added to set S.

The result of test case 2 is also pass and hence all t-way combinations {(a = 1), (b = 1), (c = 0), (d = 1), (a = 1, b = 1), (a = 1, c = 0), (a = 1, d = 1), (b = 1, c = 0), (b = 1, d = 1), (c = 0, d = 1), (a = 1, b = 1, c = 0), (a = 1, b = 1, d = 1), (a = 1, c = 0, d = 1), (b = 1, c = 0, d = 1), (a = 1, b = 1, c = 0, d = 1)} are tagged as n-FIC's and added to set S.

The result of test case 3 is also pass and hence all the t-way combinations {(a = 2), (b = 3), (c = 1), (d = 0), (a = 2, b = 3), (a = 2, c = 1), (a = 2, d = 0), (b = 3, c = 1), (b = 3, d = 0), (c = 1, d = 0), (a = 2, b = 3, c = 1), (a = 2, b = 3, d = 0), (a = 2, c = 1, d = 0), (b = 3, c = 1, d = 0), (a = 2, b = 3, c = 1, d = 0)} are tagged as n-FIC's and added to set S.

3.3 Step 3 – Determination of FIC's from LFICs:

In this step, we determine the FICs from the suspicious list. To identify the FIC from the suspicious list we generate additional TCs for each combination in the list as follows.

For each LFIC in the suspicious list, an additional TC "f" is generated, in such a way that it contains the LFIC and the probability that the TC will fail is minimum. To find the probability we formulated the equation (1) similar to the one in BEN [9].

The process to generate the additional test case is as follows.

First, generate a base TC f as follows:

For every one parameter that is not involved in LFIC, select a parameter value such that its suspiciousness value ρ is minimum.

If there exists more than one value with minimum ρ, select one randomly.

Next, verify if the base TC f is valid and the TC has never been executed before.

If it has been executed, then the TC f is returned as a new TC that contains LFIC and has minimum ρ.

If that is not the case, then pick one random parameter and change its value to a value equal to the next minimum ρ. Check this TC again to see whether it is a valid and a new TC.

Repeat these steps until a new, valid TC is found.

$$\rho(o) = \frac{1}{3}(u(o) + v(o) + w(o)) \qquad \textbf{(1)}$$

Where,

$$u(o) = \frac{number\ of\ failed\ TCs\ o\ has\ appeared}{total\ number\ of\ failed\ TCs}$$

$$v(o) = \frac{number\ of\ failed\ TCs\ o\ has\ appeared}{total\ number\ of\ TCs\ o\ has\ appeared}$$

$$w(o) = \frac{number\ of\ suspicious\ combinations\ with\ o}{total\ number\ of\ suspicious\ combinations}$$

We then execute the generated additional test case and if it passes, the corresponding LFIC is removed from the suspicious list and tagged as n-FIC. If it fails then the failure is due to this LFIC since probability that the TC f fails is minimized. Therefore, this LFIC is now tagged as FIC.

When a combination τ is marked as a F-IC, every combination "s" in the suspicious list which contains LFIC is removed because the failure is caused by this LFIC. For the example considered, the result of test case 4 is fail and the combinations {(a=1), (b=3), (c=1) , (d=0), (b=3,c=1), (a=1,c=1), (c=1,d=0), (a=1,d=0), (b=3,c=1), (a=1,c=1,d=0), (b=3,c=1,d=0)} are already marked as n-FIC's in S, therefore they cannot be the FIC's for the given test case. However, the combinations {(a=1,b=3), (a=1,b=3,c=1), (a=1,b=3,d=0) ,(a=1,b=3,c=1,d=0)} are not in S and hence tagged as LFICs and added to the suspicious list.

For the combination (a = 1, b = 3) to generate additional test case we calculate ρ values for c = 0, c = 1 and d = 0, d = 1.

$$\rho(c = 0) = \frac{1}{3}\left(\frac{0}{1} + \frac{0}{1} + \frac{0}{4}\right)$$

$$\rho(c = 1) = \frac{1}{3}\left(\frac{1}{1} + \frac{0}{3} + \frac{2}{4}\right)$$

$$\rho(d = 0) = \frac{1}{3}\left(\frac{1}{1} + \frac{1}{3} + \frac{2}{4}\right)$$

$$\rho(d = 1) = \frac{1}{3}\left(\frac{0}{1} + \frac{0}{1} + \frac{0}{4}\right)$$

$$\rho(c = 0) = \frac{1}{3}\left(\frac{0}{1} + \frac{0}{4} + \frac{0}{4}\right)$$

As $\rho(c = 0) = 0$ and $\rho(d = 1) = 0$

The new test case generated would be (a = 1, b = 3, c = 0, d = 1) and (a = 1, b = 3, c = 0, d = 1) fails. Therefore, the suspicious combination (a = 1, b = 3) is pinned down as a FIC. As all the combinations (a = 1, b = 3, c = 1), (a = 1, b = 3, d = 0) , (a = 1, b = 3, c = 1, d = 0) contain this combination they are removed from the suspicious list and pinned down as a FIC.

As (a = 1, b = 3) was a subset of all the likely F-ICs in the test case 4 it is identified as the FIC for TC 4.

4 EXPERIMENT

The proposed approach has been implemented in python and experimented on two synthetic programs. The TCs for both the programs were generated using the ACTS [10] tool. The first is the 'Account Program' taken from Software-Artifact Infrastructure Repository (SIR) [11] which deals with computing the balance in an account. The

second is a web application for selling and buying cars obtained from internet sources. The details of the 'Account Program' case study is as follows.

Account Program Case-study:

The account program is given in Fig.2.

The parameters to the program are P={a, b, c, d} and domain of the parameters are as follows:

$d_a = \{0, 1\}$, $d_b = \{0, 1\}$, $d_c = \{0, 1, 2\}$, and $d_d = \{0, 1, 2, 3\}$.

Faults were seeded to the program such that the program would reach a faulty statement when the values are as given:

- a = 0 and c = 0
- a = 1, c = 2 and d = 0
- a = 0 and d = 3

The details of the 'Car-sales ' case study is as follows.

Car-sales Case- study:

Car-sales is a web application for buying and selling cars. The web application has the parameters are P = {order-category, location, brand, Registration-number, Order-type} the values for the parameters are order-category = {buy, sell} location = {san-francisco, new-york} Brand = {BMW, Mercedes, Audi} Registration-number = {valid, invalid}, order-type = {store, online}

Fig 2: Account Program

The test case contains the values for parameters in P. Faults are seeded such that the results is faulty when:

- Order-category = 'Buy' and location = 'New-York'
- Location = 'San-Francisco' and registration = 'Valid'
- Order-category = 'Sell' and brand = 'Mercedes' and order-type = 'Store'

The original program without any faults was executed for all the possible TCs generated by ACTS. The output of the original program was compared with the output generated by the faulty program using the test oracle. The test oracle was used to build the test execution file of the faulty program. This file was used to find the FICs using the tool. The results on these two case-studies are discussed in the next section.

5 RESULTS AND DISCUSSION

The results for the 'Account' program are shown in Table-2. The column 1 shows the failed TCs for the program, the column 2 shows the suspicious combinations identified for each test case and column 3 shows the additional TCs generated for each test case to find the FICs.

As shown in row 1 of Table-2, for the test case (0, 0, 0, 0) the suspicious combinations identified are ('a', 'c'): ('0', '0'),('a', 'b', 'c'): ('0', '0', '0'), ('a', 'c', 'd'): ('0', '0', '0'),('a', 'b', 'c', 'd'): ('0', '0', '0', '0').

For the combination ('a', 'c'):(0, 0) an additional test case is generated with value 'b'=1 and d='2'. This is the test case with the least possibility of failure because the suspiciousness probability is least for 'b=1' and 'd=2'. Thus, the additional test case obtained is (0,1,0,2). Further, this test case also fails when the program is executed. Hence the combination ('a', 'c') : (0, 0) is identified as a FIC. The combination ('a', 'c'):(0, 0) is a subset of all the members in the suspicious set, hence the combination ('a', 'c'):(0, 0) is marked as the minimal-FIC for the test case (0, 0, 0, 0) and all the members are deleted from the suspicious list as they are not minimal FIC.

For the test case (0, 1, 0, 0) as the combination ('a', 'c') : (0, 0) was identified as a minimal-FIC, the suspicious set is empty. Similarly, FICs are identified for each failed test case as shown from row 2 to row 18 in Table-2

The failed TCs for the Car-Sales application are shown in Table-3. FICs are identified for each TC and the steps to find the FICs for one of the TCs is as follows.

For the test case ('Buy', 'New-York', 'Mercedes', 'Invalid', 'Online'):

The suspicious combinations generated are ('Order-Category', 'Location'): ('Buy', 'New-York'), ('Order-Category', 'Location', 'Brand'): ('Buy', 'New-York', 'Audi'), ('Order-Category', 'Location', 'Reg-No'): ('Buy', 'New-York', 'Invalid'), ('Order-Category', 'Location', 'Order'): ('Buy', 'New-York', 'Online'), ('Order-Category', 'Location', 'Brand', 'Reg-No'): ('Buy', 'New-York', 'Audi', 'Invalid'), ('Order-Category', 'Location', 'Brand', 'Order'): ('Buy', 'New-York', 'Audi', 'Online'), ('Order-Category', 'Location', 'Reg-No', 'Order'): ('Buy', 'New-York', 'Invalid', 'Online').

Initially for the Suspicious combination ('Order-Category', 'Location'): ('Buy', 'New-York') a new test case is generated with brand = BMW, Registration-no = invalid, order-type = online. This test case also fails, hence the combination ('Order-Category', 'Location'): ('Buy', 'New-York') is marked as FIC. As all the members in the suspicious list contain ('Order-Category', 'Location'): ('Buy', 'New-York'), all of them are marked as FIC and are deleted from the suspicious list. Similarly FICs are identified for all the failed TCs , the results are tabulated in Table-3.

Failed Test case				Suspicious Combinations	Additional test cases				Failure Inducing Combination
a	b	c	d		a	b	c	d	
0	0	0	0	('a', 'c'): ('0', '0'), ('a', 'b', 'c'): ('0', '0', '0'), ('a', 'c', 'd'): ('0', '0', '0'), ('a', 'b', 'c', 'd'): ('0', '0', '0', '0')	0	1	0	2	('a', 'c'): ('0', '0')
0	1	0	0	-	-	-	-	-	('a', 'c'): ('0', '0')
1	0	2	0	('a', 'c', 'd'): ('1', '2', '0'), ('a', 'b', 'c', 'd'): ('1', '0', '2', '0')	1	1	2	0	('a', 'c', 'd'): ('1', '2', '0')
1	1	2	0	-	-	-	-	-	('a', 'c', 'd'): ('1', '2', '0')
0	0	0	1	-	-	-	-	-	('a', 'c'): ('0', '0')
0	1	0	1	-	-	-	-	-	('a', 'c'): ('0', '0')
0	0	0	2	-	-	-	-	-	('a', 'c'): ('0', '0')
0	1	0	2	-	-	-	-	-	('a', 'c'): ('0', '0')
0	0	0	3	('a', 'd'): ('0', '3'), ('a', 'b', 'd'): ('0', '0', '3') ('a', 'b', 'c', 'd'): ('0', '0', '0', '3')	0	1	2	3	('a', 'd'): ('0', '3'), ('a', 'c'): ('0', '0')
0	1	0	3	-	-	-	-	-	('a', 'd'): ('0', '3'), ('a', 'c'): ('0', '0')
0	0	1	3	-	-	-	-	-	('a', 'd'): ('0', '3')
0	1	1	3	-	-	-	-	-	('a', 'd'): ('0', '3')
0	0	2	3	-	-	-	-	-	('a', 'd'): ('0', '3')
0	1	2	3	-	-	-	-	-	('a', 'd'): ('0', '3')

Table 2: Results of Account program

Failed Test case					# suspicious combinations	# additional test cases	Failure Inducing Combination
Order-category	Location	Brand	Reg No	Order Type			
Buy	N-Y	Merc	Valid	Online	11	1	(order-category, Brand):(Buy, Merc)
Buy	N-Y	Merc	Valid	Store	0	0	(order-category, Brand):(Buy, Merc)
Buy	N-Y	Merc	Invalid	Online	0	0	(order-category, Brand):(Buy, Merc)
Buy	S-F	Merc	Valid	Online	9	1	(Location, Reg No):(S-F, Valid)
Sell	N-Y	Merc	Valid	Store	7	2	(order-category, Brand, Order-Type): (Buy, Merc, Store)

Table 3: Finding FICs for Web Application

6 CONCLUSION

The current paper works with an existing executed test suite as input to identify the combinations of parameters that caused the fault to localize it. The discussed approach automatically identifies the underlying FIC irrespective of whether it is a 2-way, 3-way or t-way.; whereas in BEN it has to be tried starting with 2-way and going all the way up to n-way for a specific failed test case till the FIC is uniquely found.

The approach has been tried on synthetic case-studies and found to be working well. In future, we plan to apply it on real-life test suites of projects in the industry. The approach works well for small TCs which have up to 5 parameters. Scalability to more parameters is yet to be determined.

REFERENCES

[1] Z. Zhang, and J. Zhang. “Characterizing failure-causing parameter interactions by adaptive testing”, In Proceedings of ACM International Symposium on Software Testing and Analysis (ISSTA 2011), pp. 331-341, 2011.

[2] J. Li; C. Nie, and Y. Lei, "Improved Delta Debugging Based on Combinatorial Testing," In Proceedings of International Conference on Quality Software (QSIC), pp.102,105, 2012.

[3] Zeller and R. Hildebrandt. “Simplifying and isolating failure inducing input”, In Proceedings of the IEEE Transactions on Software Engineering, 2002, pages 183–200.

[4] Sekaran, R., Munnangi, A. K., Ramachandran, M., & Khishe, M. (2025). Cayley–Purser secured communication and jackknife correlative classification for COVID patient data analysis. Scientific Reports, 15(1), 4666.

[5] Z. Wang, B. Xu, L. Chen, and L. Xu. “Adaptive interaction fault location based on combinatorial testing”, In Proceedings of the 10th International Conference on Quality Software (QSIC 2010), pages 495–502, 2010.

[6] C. Nie, H. Leung, and B. Xu. “The minimal failure-causing schema of combinatorial testing”, In ACM Transactions on Software Engineering and Methodology, 20(4) , September 2011.

[7] C. Yilmaz, M. B. Cohen, A. A. Porter. “Covering arrays for efficient fault characterization in complex configuration spaces”, In Proceedings of the IEEE Transaction on Software Engineering, 2006, 32(1): 20-34.

[8] K. Shakya, T. Xie, N. Li, Y. Lei, R. Kacker, and R. Kuhn, "Isolating Failure-Inducing Combinations in Combinatorial Testing Using Test Augmentation and Classification," In proceedings of 5th IEEE International Conference on Software Testing, Verification and Validation (ICST), pp.620-623, 2012.

[9] Ghandehari, Laleh Sh, Yu Lei, Raghu Kacker, D. Richard, Rick Kuhn, David Kung, and Tao Xie. "A Combinatorial Testing-Based Approach to Fault Localization." IEEE Transactions on Software Engineering (2018).

[10] Advanced Combinatorial Testing System (ACTS), http://csrc.nist.gov/groups/SNS/acts/documents/comparisonreport.html, 2015.

[11] Software-artifact Infrastructure Repository, http://sir.unl.edu/portal/index.php, 2012.

[12] Jayaram, Rekha, and R. Krishnan. "Approaches to fault localization in combinatorial testing: A survey." Smart Computing and Informatics: Proceedings of the First International Conference on SCI 2016, Volume 2. Springer Singapore, 2018.

[13] Ghandehari, Laleh Sh, et al. "BEN: A combinatorial testing-based fault localization tool." 2015 IEEE Eighth International Conference on Software Testing, Verification and Validation Workshops (ICSTW). IEEE, 2015.

[14] Fadhil, Heba Mohammed, Mohammed Najm Abdullah, and Mohammed Issam Younis. "Combinatorial Testing Approaches: A Systematic Review." IRAQI JOURNAL OF COMPUTERS, COMMUNICATIONS, CONTROL AND SYSTEMS ENGINEERING 22.4 (2022): 60-79.

[15] Arcaini, Paolo, Angelo Gargantini, and Marco Radavelli. "Efficient and guaranteed detection of t-way failure-inducing combinations." 2019 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW). IEEE, 2019.

[16] Sekaran, R., Munnangi, A. K., Ramachandran, M., & Gandomi, A. H. (2022). 3D brain slice classification and feature extraction using Deformable Hierarchical Heuristic Model. Computers in Biology and Medicine, 149, 105990.

[17] Tatale, Subhash, and V. Chandra Prakash. "Enhancing acceptance test driven development model with combinatorial logic." International Journal of Advanced Computer Science and Applications 11.10 (2020).

[18] Friedrichs, Torben, Konrad Fögen, and Horst Lichter. "A comparison infrastructure for fault characterization algorithms." 2020 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW). IEEE, 2020.

[19] Jayaram, Rekha, and R. Krishnan. "Combinatorial Test Perspective on Test Design of SDN Controllers." Proceedings of Second International Conference on Sustainable Expert Systems: ICSES 2021. Singapore: Springer Nature Singapore, 2022.