International Journal of Scientific Research in Computer Science, Engineering and Information Technology
© 2018 IJSRCSEIT | Volume 3 | Issue 3 | ISSN : 2456-3307

Fault Locating and Identifying with minimal Interaction in Testing of Software Engineering Applications

Muzammil H Mohammed[1],Faiz Baothman[2]
[1]Department of Information Technology College of Computers and Information Technology, Taif University, Taif, Saudi Arabia
[2]Department of Computer Science College of Computers and Information Technology, Taif University, Taif, Saudi Arabia

ABSTRACT

Combinatorial testing method may additionally substantially cut again checking out fee and increase software program exceptional. By victimisation at suite generated with the aid of Combinatorial testing as input to conduct black-container trying out towards a machine, at some stage in a action at regulation, there may also be entirely part of all its parameters relevant to the defects in system and consequently the interaction by using the ones partial parameters is critical trouble of triggering fault. If we can discover those parameters appropriately, this could facilitate the software program package deal and checking out technique. This paper proposes a completely unique algorithmic program named Fault Interaction Location to find those interactions that cause device's disasters and intervening time By applying this technique, testers will analyze and locate the factors applicable to defects, so creating the approach of software program package trying out and debugging simpler and further low-budget. The consequences of our study suggest that Fault Interaction Location plays better compared with fault place techniques in combinatorial trying out due to its advanced effectiveness and exactness.

Keywords: Test Suite**,** localization, Detection, Arrays, Debugging, Fault Interaction, Algorithm

I. INTRODUCTION

Combinatorial checking might considerably scale back test price and increase quality of code [1]. it's been verified to be effective particularly during a code wherever faults return from the interactions of its parameters [2]. Combinatorial testing might observe the parameter interactions that trigger the faults instead of localize it. If a test suit triggers the fault of a system, it reflects that there exists one or a lot of defects within the program [3–5]. However, not all parameters within the test suit square measure relevant to defects. If we tend to square measure able to find a parameter within the test suit that's relevant to the fault, we are able to apply this handy data to facilitate the debugging method.

In combinatorial testing, the study on fault interaction technique might be categorised into adaptive technique and maladaptive technique consistent with the dependence between further check cases and running results [6, 7]. For maladaptive ways, the generation of further check cases doesn't trust the running results of original check cases. Colbourn and McClary [8] gift a maladaptive technique named Locating and Detecting Arrays (LDA). supported basic celebrated data like parameters' range, values, and faults' range, the strategy applies t-way Locating and Detecting Array to find faults in code. Mart´ınez et al. [9] gift a self-adaptive formula supported Errors Locating

Arrays (ELA) and analyze the formula quality. However, this technique might solely be used underneath the condition that the value's range of every parameter in software package isn't larger than two. Hagar et al. [1] propose the strategy of Partial Covering Array (PCA) that might be employed in the software package with celebrated safe price, and it presents a replacement combinatorial structure to get group. Another class is thought as adaptive technique [10],whose generation of further check cases depends on the data given by the execution of original check cases .Zeller and Hildebrandt [11] gift a typical adaptive technique named Delta Debugging. This technique is to spot the interaction that's relevant to the faults by modifying the input parameters. For a test suit that triggers the fault, modify a number of its input parameters; if the changed test suit still triggers the fault, then the changed parameters square measure extraneous to fault; otherwise, the changed parameters square measure associated with fault. supported Delta Debugging, Z. Zhang and J. Zhang [12] gift a technique named FIC. like Delta Debugging, FIC modify one parameter during a test suit with n parameters once. Then repeat this method n times and also the token fault interaction is calculated subsequently. A restraint of Delta Debugging primarily based ways is that they might solely be applicable to the test suit containing one token fault interaction, First, when the execution of original check cases, the check cases are divided into two sets: FTS and PTS; the previous contains check cases that trigger faults, whereas the latter contains check cases that don't trigger faults. Then we tend to determine the set of interactions lined by FTS however not lined by PTS and name this set as candidate faulty interaction set (canFIS). Second, we tend to generate further check cases to pick interactions in canFIS so the token fault interaction set is obtained finally.

II. THEOREMS

Assume there's a System Under Test (SUT) with n input parameters as P = one, $P2$, $P3$, . . . , Pn}. For any $i \in [1,]$, the vary of parameter P i is denoted as D i. And combine $\langle Pi, V\rangle$ is employed to indicate that the worth of parameter Pi is V, additionally denoted as $(Pi.V)$.In this model, the vary of parameter P a pair of is D a pair of =. The input parameter model of SUT is denoted as SUT(n : ($|D|$, $|D2|$, $|D3|$, . . . , $|Dn|$)),which means SUT has n input parameters and, for every parameter $P1$, $P2$, $P3$, . . . , Pn, the worth numbers are$|D1|$, $|D2|$, $|D3|$, . . . , $|Dn|$, severally. For a group of t elements$I$ = , if for any m, s, t, 1 ≤m, $s \le t \le n$, there area unit is, $im\in$; ($m \neq s$, $im\neq is$), then we tend to decision I a t-way interaction.

Theorem 1. for two interactions $I1$ and $I2$, if one ⊂ $I2$, then $subSet(I1) \subset subSet(I$ a pair of Proof. From Definition three (subinteraction set), we are able to grasp that $i \neq \Phi$ and for any $i \in$ subSet($I1$), $i \subset I1$, therefore $i \subset I2$; specifically, subSet($I1$) ⊂ subSet($I2$).
Theorem 2. Divide (m, t; ($P1$, $P2$, $P3$, . . . , Pn)) into a pair of sets:FTS and PTS. In FTS, all check cases triggered the fault of the system, PTS = (m, t; ($P1$, $P2$, $P3$, . . . , Pn)) – FTS; if I is that the nominal fault interaction of CA, then $I \subset canFIS$.
Proof. From abstract thought one we are able to grasp that, for any nominal fault interaction i in (m, t; ($P1$, $P2$, $P3$, ., Pn)), there's a failing action T that satisfies $i \in$ sunSet(T). therefore $i \in \cup T \in$FTS subSet(T) and $i \in \cup T \in$PTS subSet(T); that's, $i \in$canFIS..
Theorem 3. Use MFIS that denotes all nominal fault interactions set of (m, t; ($P1$, $P2$, . . . , Pn)); for a nominal fault interaction i, if $i \in canFIS$ and $R(addTF(i)) = pass$,then $MFIS \cap \cup I \in i\ subSet(I)$ = zero and $MFIS \subset canFIS - \cup I \in i\ subSet(I)$.
Proof. this might be directly over from Theorem a pair of.
Theorem 4. For a t-way interaction I, the extra action is T = (I). In subinteraction set of T, the amount of interactions that don't belong to (I) is $2n$–$2t$.
Proof. From Definition four we all know that |subSet(T)| = $2n$– one and |subSet(I)| = $2t$– one, within which I is that the sub interaction of T. From

Theorem one we all know that subSet(I) ⊂ subSet(T). So |subSet(I) – subSet(T)| = 2n– 2t.

III. FAULT INTERACTION LOCATION ALGORITHM

3.1. Description of FIL. Theorem three and Definition five give a screening technique for getting the set of all token fault interactions from the set of candidate fault interactions; once an interaction I proves to be the fault interaction, we delete all the parent interactions of I from canFIS except I, whereas we have a tendency to delete all the kid interactions of I from canFIS except I once it proves to not be the fault interaction. It has 2 input parameters CA ANd (T) and an output parameter canFIS that is that the token fault interaction set. usually the formula method can be divided into a pair of phases. Generate the canFIS for fault location. Firstly, the formula counts the amount of times that the fault interaction exists in passed or unsuccessful actions on an individual basis (steps (2) to (12)); then the canFIS is screened out from take a look at case set CA (steps (13) to (15)).Generate the token fault interaction set (steps (16) to (20)). Steps (17) and (18) describe the following: if a schema I may be a fault interaction, then we have a tendency to delete all its super interaction apart from I in canFIS. Steps (19) ANd (20) describe the following: if an interaction isn't a fault interaction, then we have a tendency to delete all components in its sub interaction in canFIS.

(1) Implementation of set. The input of perform set is AN interaction I, and its output is that the subinteraction set of I. For n-way interaction In= , its t-way subinteraction may be a binary string (b1, b2, b3, . . . , bn), within which Σni=1bi= t; that's, a t-way subinteraction of In is = Vij∈ In∧ bij= 1, within which i1, i2, i3, . . . , it don't seem to be adequate to one another.

From Theorem four we all know that once generating addition action T, we'll 1st make sure that there's no token fault interaction in 2n– 2t subinteractions. However, if n is extremely massive and t is comparatively tiny, the numbers of interactions to be examined and extra take a look at cases to be generated ar terribly massive. For convenience, we have a tendency to assume every parameter features a worth, that doesn't go with any fault (i.e., this worth doesn't belong to any token fault interaction).This worth is denoted as safe worth. The number of times that every interaction seems in passed take a look at cases and unsuccessful take a look at cases has to be recorded. For a unidirectional interaction, that is, a worth of a parameter, the worth $f/(f + p)$ is known as the fault quantitative relation of the worth. We merely think about that the smaller the fault quantitative relation of a parameter's worth is, the additional probably this worth would be the safe worth of the parameter.

ALGORITHM: The FIL algorithm

Inputs: CA: test case set
(T): test result

Output: canFIS: the minimal fault interaction set of CA.

Process:

(1) Set canFIS = Φ, AllSet = Φ
//canFIS save Candidate Fault Interaction Set;
//AllSet save elements of CA, each element has 2 properties p, f.
//phase I, Generate Candidate Fault Interaction Set
(2) for(eachtest cases T in CA){
(3) if($R(T)$ == fail){

```
(4)             for (each element i in subSet(T)){
                                    //subSet() is a key function, we will explain it later.
(5)               if (i ∉ AllSet){
(6)                   AllSet = AllSet ∪ {i}
                      }
(7)                   AllSet[i].f + +
                      //f is the number of test cases which include interaction and triggered system fault;
                  }
(8)     }else{
(9)             for (each_ element i in subSet(T)){
(10)                  if (i ∉ AllSet){
(11)                      AllSet = AllSet ∪ {i}
                      }
(12)                  AllSet[i].p + +
                      //p is the number of test cases which include interaction but not trigger system fault;
                  }
        }
(13)    for(each_element i in AllSet){
(14)      if (i.p == 0){
(15)          canFIS = canFIS ∪ {i}
              }
        }
        //phase II, Generate the minimal fault interaction set
(16)    while(there_are_ element i not tested in canFIS){
(17)          if (R(addTF(P1.IE)) == fail){
                  //addTF() is another key function, we will implement it later.
(18)              canFIS = canFIS – {I | I ∉ canFIS, i ⊆ I}
(19)          }else{
(20)              canFIS = canFIS – subSet(I)
              }
        }
(21)    return canFIS
```

First, if a value of a parameter ($1 \le i \le n$) appears in interaction I, the value of pi in additional test case T is pi itself; otherwise it will be assigned by the value that has the smallest safe value. If many values of pi have the same smallest safe value, then pi will be assigned randomly among these values.

Second, to check T, if it does not belong to test suite CA, then T is used as an additional test case; otherwise, T will be regenerated. The regeneration process is to modify a parameter's value in T by assigning this parameter another value whose safe value is the smallest or second smallest, thus making sure I is a sub interaction of T. Then we repeat this process till T does not belong to CA.

Algorithm 2: A Fault algorithm

```
(1)  public static int foo (int a, int b, int c, int d)
(2)  {
(3)      int r = 1;
(4)      b+ = a + c;
(5)      switch (a)
(6)      {
(7)              case 0:
(8)                      if (c < 1‖d > 2)
(9)
(10)                 //should be: r+ = (b – d)/(a+ 2);
(11)                         r = (b – d)/(a+ 2);
(12)                 else
(13)                         r = b/(c + 2);
(14)                 break;
(15)             case 1:
(16)                     r = c ∗ (a –d);
(17)                 break;
(18)     }
(19)     return r;
(20) }
```

Table 2 Test Result of 2-Way Coverage

Test#	a	b	c	d	Result
1	0	0	0	0	Fail
2	1	1	1	0	Pass
3	0	1	2	0	Pass
4	1	0	0	1	Pass
5	0	0	1	1	Pass
6	1	1	2	1	pass
7	0	1	0	2	Fail
8	1	0	1	2	Pass
9	0	0	2	2	Pass
10	0	1	0	3	Fail
11	1	0	1	3	Pass
12	1	0	2	3	Pass

Table 3. The canFIS of CA

Test#	a	b	c	d	Result
1	0	0	0	0	Fail
2	1	1	1	0	Pass
3	0	1	2	0	Pass
4	1	0	0	1	Pass
5	0	0	1	1	Pass
6	1	1	2	1	pass
7	0	1	0	2	Fail
8	1	0	1	2	Pass
9	0	0	2	2	Pass
10	0	1	0	3	Fail
11	1	0	1	3	Pass
12	1	0	2	3	Pass
13	0	0	0	0	Fail
14	1	1	1	0	Pass

15	0	1	2	0	Pass
16	1	0	0	1	Pass
17	0	0	1	1	Pass
18	1	1	2	1	pass
19	0	1	0	2	Fail
20	1	0	1	2	Pass
21	0	0	2	2	Pass
22	0	1	0	3	Fail
23	1	0	1	3	Pass

Table 3 shows the results of first process, in which the first column refers to the number of interactions while the first row represents the parameter of each interaction and the remaining rows represent the respective value of the parameters. Each row in Table 3 indicates an interaction from row 2 on. For example, interaction 1 {a.0, b.0, c.0} is shown in row 2 (I # 1).The second process is to generate minimal fault interaction set, as shown in Table 4. The second column of Table 4describes the interactions contained by canFIS in each step.The third column in Table 4 shows the interactions under testing. The fourth column shows the additional test cases for under testing interactions. Columns 5 and 6 show the outputs of additional test cases and the set consisting of the elements deleted from canFIS, respectively.

Table 4. The step of computing canFIS.

Step #	canFIS	I	T= addTF(i)	R(T)	Delete from canFIS
(1)	{1, 2, . . . ,23}	1	(0, 0, 0, 1)	Fail	{8}
(2)	{1, 2, ... 7, 9, 10, . . . , 23}	2	(0, 0, 1, 0)	Pass	{2, 6}
(3)	{1, 3, 4, 5, 7, 9, 10, . . . , 23}	3	(0, 1, 0, 0)	Fail	0
(4)	{1, 3, 4, 5, 7, 9, 10, . . . , 23}	4	(1, 0, 0, 0)	Pass	{4, 7}
(5)	{1, 3, 5, 9, 10, . . . , 23}	5	(0, 1, 0, 1)	Fail	{1, 3, 9, 10, 12, 17, 18}
(6)	{5, 11, 13, 14, 15, 16, 19, 20, 21, 22, 23}	11	(1, 1, 0, 1)	Pass	{11}
(7)	{5, 13, 14, 15, 16, 19, 20, 21, 22, 23}	13	(1, 1, 0, 2)	Pass	{13}
(8)	{5, 14, 15, 16, 19, 20, 21, 22, 23}	14	(1, 1, 1, 2)	Pass	{14}
(9)	{5, 15, 16, 19, 20, 21, 22, 23}	15	(1, 1, 0, 2)	Pass	{15}
(10)	{5, 16, 19, 20, 21, 22, 23}	16	(0, 1, 1, 2)	Pass	{16}
(11)	{5, 19, 20, 21, 22, 23}	19	(1, 1, 2, 3)	Pass	{19}
(12)	{5, 20, 21, 22, 23}	20	(0, 1, 1, 3)	Fail	{23}
(13)	{5, 20, 21, 22}	21	(1, 1, 2, 3)	Pass	{21}
(14)	{5, 20, 22}	22	(1, 1, 0, 3)	Pass	{22}
(15)	{5, 20}	—	—	—	—
(16)	{{a.0, c.0}, {a.0,d.3}}	—	—	—	—

We can conclude from Table 4 that the whole process takes 14 steps and each step generates an additional test case. The minimal fault interaction set {5, 20} is screened out at step (15) at last. Meanwhile we can get a conclusion that the number of steps the process takes depends on the order of interactions being tested. For example, if, in step (13) test interaction 22, the element to delete in canFIS is {21,

22},then the minimal fault interaction set could be generated directly. The whole process only takes 13 steps and needs only 13 test cases. Therefore, an optimized interaction test order could reduce the generating of test cases. In this paper, we do not carry on the discussion and simply consider the order is random. The method presented by Ghandehari et al. [13] give the result that contains 9 interactions in the set; however, the minimal fault interaction set contains 2 interactions. This shows that the method comFIL is more precise than the method proposed by Ghandehari et al.

IV. EXPERIMENT AND TESTING

Evaluation Criteria. We use fault quantitative relation as our analysis criteria. In FIL algorithmic rule, we'd like to record range |the amount|the quantity} p of every interaction that exists in passed take a look at cases and therefore the number f of every interaction that exists in unsuccessful take a look at cases. For a 1-way interaction, the worth of $f/(f + p)$ is termed fault quantitative relation. The smaller the worth is, the additional probable worth|the worth} are going to be a secure value of the parameter. Since the feature of those programs isn't a priority during this paper, they're assumed to be correct. Then the quality and fault versions are compiled and run with action T as input; if the outputs of ordinary and fault versions ar totally different, we have a tendency to believe the action triggers the fault; that's, $R(T)$ = fail; otherwise, $R(T)$ = pass.we have a tendency to use six C programs (comdline,count, nametbl, ntree, series, and tokens [12]) as take a look at samples and input parameter model conferred by Z. Zhang and J. Zhang [12]. Table five shows the essential info of those programs.The second column represents the amount of lines while not comments in every program,while column3 refers to their input models. for instance, comdline (9; (21, 34, 41, 62, 151))means comdline has nine parameters, within which four parameters have three values, a pair of parameters have only one worth, one parameter has a pair of values, one parameter has four values, a pair of parameters have half-dozen values, and one parameter has fifteen values. Count (6; (2, 2, 3, 3, 3, 3)) may also be painted as count (6; (22, 34))

Table 5: Test sample.

Program	Number of lines	Input model
Comdline	42	Comdline (9;(2^1, 3^4, 4^1, 6^2, 15^1))
Count	288	Count(6;(22, 34))
Nametbl	129	Nametbl (8; (24, 32, 52))
Ntree	307	Ntree(6; (22, 44))
Series	329	Series (4; (21, 42, 61))
Tokens	336	Tokens (4; (22, 32))

V. RESULT

Table 6 indicates the careful check results of the experiment; the info is principally targeted on check steps (additional check cases) and base. In Table 6, column a pair of shows the quantity of check cases. Column three represents the various fault versions of every program. Column four shows the scale of every canFIS. Column five refers to the quantity of check cases required by every canFIS. Columns 6-11 represent range|the amount|the quantity} of x-way marginal fault interactions and therefore the number of x-way fault interactions to be hand-picked, severally; x can be known by the column title. as an example, the primary fault version of comdline is shown in Table half dozen. It means its canFIS's size is 1663, and it desires 149 check cases. The numbers of 1-5-way marginal fault interactions square measure 1; zero; 0; 0; 0, severally, and therefore the marginal fault interaction larger than five ways in

which is zero.The numbers of interactions being tested for computing every canFIS square measure 1; 15; 20; 22; 27; sixty four, respectively. The simulated experiment result bestowed by Figures one and a couple of shows the quantity of further check cases is decreasing whereas the program may higher match the three assumptions of this paper. moreover, with the increment of marginal fault interactions' count, the magnitude relation becomes smaller; this implies the upper likelihood of obtaining safe price for input parameters. However, once the marginal fault interaction will increase to a special price like nine in Figure 2(b), the magnitude relation would rise; this can be as a result of once the quantity of input parameters becomes large, too several further check cases are going to be generated; this may have an effect on the potency and lead the magnitude relation to rise.

Table 6. Test result of standard program.

Program	CA size	ver	canFIS radix	canFIS steps	1	2	3	4	5	Over 5
Comdline	95	1	1663	149	1 15	0 20	0 22	0 27	0 64	0
		2	10734	2033	0 0 25	0 6 29	50 219 87	28 415 24	0 613 824	0 780 0
		3	2392	284	1	25	29	87	24	82
Count	12	1	121	36	0 0	2 2	0 19	0 14	0 1	0 0
		2	250	89	0 0	4 5	0 41	0 24	0 1 9	0 0
Nametbl	25	1	23	12	0 0	0 1	2 6	0 5	0 0	- -
		2	2	109	0 54	7 0	4 9	0 31	0 14	- 0 -
		3	41	20	0 0	0 0	3 9	0 11	0 0	- -
NTree	16	1	47	43	0 0	1 32	1 11	0 0	- -	- -
		2	66	48	0 0	3 32	0 16	0 0	- -	- -
Series	24	1	18	15	0 0	0 7	2 8	0 0	- -	- -
		2	57	34	0 0	13 19	0 17	0 0	- -	- -
Tokens	3	1	23	13	0 0	0 9	0 3	0 0	- -	- -
		2	23	10	0 0	0 6	0 4	0 0	- -	- -

VI. CONCLUSION AND FUTURE WORK

6.1. Conclusion. During this paper, we have a tendency to gift a replacement combinatorial testing formula named Fault Interaction Location formula that may sort the tokenish fault interaction set of check cases. the most contributions of this paper area unit listed as follows:
(1) Summarizing the fundamental plan of the previous fault interaction location techniques, as well as their blessings and downsides.
(2) Proposing a completely unique fault interaction location technique named Fault Interaction Location formula that has additional powerful functionalities and performs additional exactly compared with different fault location techniques.

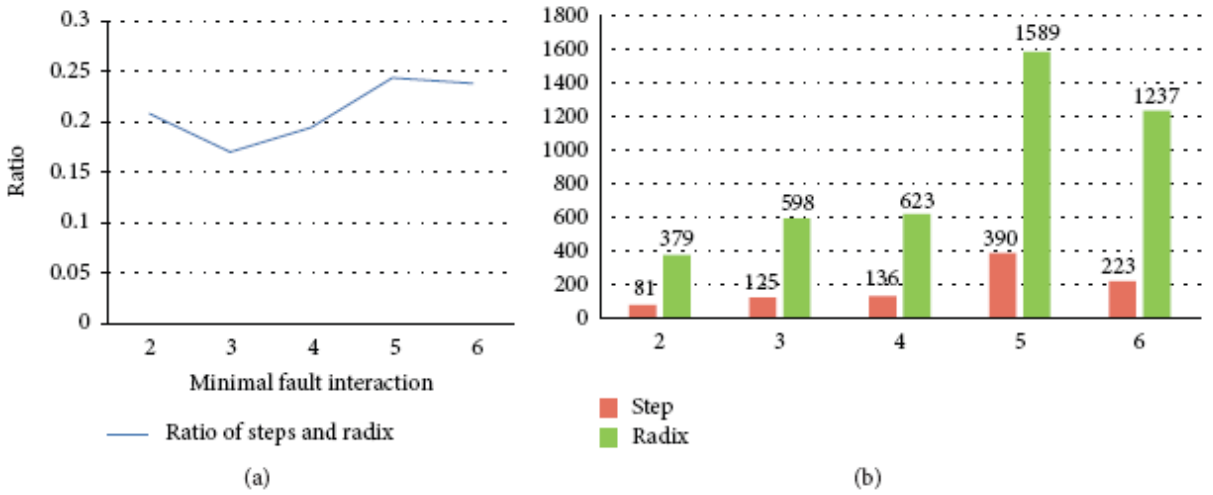

Figure 1. Test result of Simulated Experiment 1. (a) Ratio of steps and radix. (b) Step and radix.

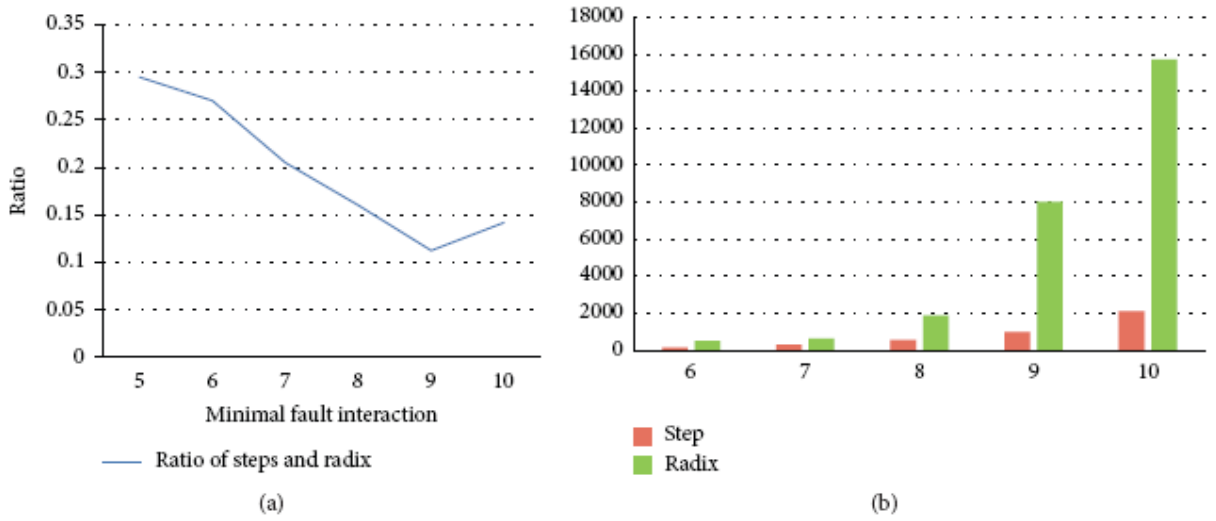

Figure 2. Test result of Simulated Experiment 2. (a) Ratio of steps and radix. (b) Step and radix

If we have a tendency to cannot get the safe worth of every parameter before testing, the value in generating an extra legal action for associate interaction is extremely high. However, if there are solely a couple of bugs in an exceedingly program, the quantity of stripped-down fault interactions is little and it's a lot of probable for a parameter to be in its safe worth.. once we calculate the safe values of every parameter with correct strategies, it's nearly not possible that generated extra take a look at cases

don't satisfy Assumption three. Even for a legal action generated for associate interaction indiscriminately, its chance that it doesn't satisfy Assumption three is quite low. therefore nearly each testing technique in combinatorial testing may solely work effectively once applied in program with comparatively less faults the speculation and experiments indicate that FIL is a lot of correct in fault localization compared with different algorithms in terms of combinatorial testing.

The future work can embody 3 aspects: uncountable extra take a look at cases ought to be generated; but, the order of the interactions being take a look ated can influence the quantity of extra test cases generated. therefore optimizing the order of interactions being take a look ated to scale back the quantity of extra test cases are going to be the target of formula is to get the stripped-down fault interaction set, whereas the way to use the stripped-down fault interaction set to additional find bugs also will be a big topic to check additional later. Exploring simpler combinatorial testing technique in step with totally different style of software package (such as net service) is additionally worthy of additional study.

VII. REFERENCES

[1]. J. D. Hagar, T. L. Wissink, D. R. Kuhn, and R. N. Kacker,"Introducing combinatorial testing in a large organization,"Computer, vol. 48, no. 4, pp. 64–72, 2015.

[2]. W.-J. Zhou, D.-P. Zhang, and B.-W. Xu, "Locating error interactions based on partial covering array," Chinese Journal of Computers, vol. 34, no. 6, pp. 1126–1136, 2011.

[3]. C. Nie and H. Leung, "The minimal failure-causing schema of combinatorial sting,"ACMTransactions onSoftwareEngineering and Methodology, vol. 20, no. 4, article 15, 2011.

[4]. B. Garn and D. E. Simos, "Eris: a tool for combinatorial testing of the Linux system call interface," in Proceedings of the 7th IEEEInternational Conference on Software Testing, Verification and Validation Workshops (ICSTW '14), pp. 58–67, Cleveland, Ohio,USA, April 2014.

[5]. S. K. Khalsa andY.Labiche, "An orchestrated survey of available algorithms and tools for combinatorial testing," in Proceedings of the 25th IEEE International Symposium on Software ReliabilityEngineering (ISSRE '14), pp. 323–334, IEEE, Naples, Italy,November 2014.

[6]. A. Gupta and C.H. Scholz, "A model of normal fault interaction based on observations and theory," Journal of Structural Geology,vol. 22, no. 7, pp. 865–879, 2000.

[7]. D. C. P. Peacock, "Propagation, interaction and linkage innormal fault systems," Earth-Science Reviews, vol. 58,no. 1-2, pp.121–142, 2002.

[8]. C. J. Colbourn and D. W. McClary, "Locating and detecting arrays for interaction faults," Journal of Combinatorial Optimization,vol. 15, no. 1, pp. 17–48, 2008.

[9]. C. Mart´ınez, L. Moura, D. Panario et al., "Algorithms to locate errors using covering arrays," in LATIN 2008:Theoretical Informatics, pp. 504–519, Springer, Berlin,Germany, 2008.

[10]. J. Piton-Gonc¸alves and S. M. Alu´ısio, "An architecture for multidimensional computer adaptive test with educational purposes," in Proceedings of the 18th Brazilian symposium onMultimedia and the web (WebMedia '12), pp. 17–24, ACM, S˜aoPaulo, Brazil, 2012.

[11]. A. Zeller and R.Hildebrandt, "Simplifying and isolating failureinducing input," IEEE Transactions on Software Engineering, vol.28, no. 2, pp. 183–200, 2002.

[12]. Z. Zhang and J. Zhang, "Characterizing failure-causing parameter interactions by adaptive testing," in Proceedings of the20th International Symposium on Software Testing and Analysis(ISSTA '11), pp. 331–341, ACM, Toronto, Canada, July 2011.

[13]. L. Ghandehari, J. Chandrasekaran, Y. Lei, R. Kacker, and D. Kuhn, "Short paper: BEN: a combinatorial testing-based fault localization tool," in Proceedings of the 4th InternationalWorkshop on Combinatorial Testing (in Junction with 8th IEEEInternational Conference on Software Testing, Verification andValidation), April 2015.

[14]. Wei Zheng, Xiaoxue Wu, Desheng Hu, and Qihai Zhu College of Software & Micro-Electronics, Northwestern Polytechnical University, Xi'an 710072, China

[15]. L. S. Ghandehari, Y. Lei, D. Kung, R. N. Kacker, and D. R. Kuhn,"Fault localization based on failure-inducing combinations," inProceedings of the 24th IEEE International Symposium on SoftwareReliability Engineering (ISSRE '13), pp. 168–177, Pasadena,Calif, USA, November 2013.

[16]. M. B. Cohen, M. B. Dwyer, and J. Shi, "Interaction testing of highly-configurable systems in the presence of constraints," inProceedings of the International Symposium on Software Testingand Analysis, pp. 129–139, ACM, 2007.