Referee: 1  
  
Comment 1:

First, on page 11, in table IX, you give the test cases generated by FIC\_BS (t1 to t4).  
Unfortunately, you don't explain how FIC\_BS chooses these test cases, and whether  
they are chosen incrementally (choose t1 then decide you need t2...) or simultaneously.  
The process of choosing the test cases should be better explained.

Response: According to this comment as well as Comment 6, we have changed the description of our approach with simpler MFS identification approach OFOT. OFOT each time mutates only one factor of the original failing test case, such that it can identify those failure-inducing factors by comparing the results of the mutated test cases with that of the original failing test case.

As to FIC\_BS, it can be regarded as the binary search version of OFOT. Specifically, OFOT successively select one factor of the original failing test case to mutated, while FIC\_BS choose half of the factors to be mutated part. Here, to mutate which half depends on the result of the last generated test case. Hence, FIC\_BS generated test cases incrementally. The processes of OFOT and FIC\_BS can depicted in the following table.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Approach |  | OFOT | |  |  | FIC\_BS | |
| Original test | T0 | 0 0 0 0 0 0 0 0 | Fail | T0 | 0 0 0 0 0 0 0 0 | Fail |
| Mutated tests | T1 | 1 0 0 0 0 0 0 0 | Fail | T1’  T2’  T3’  T4’ | 1 1 1 1 0 0 0 0 | Pass |
| T2 | 0 1 0 0 0 0 0 0 | Fail | 1 1 0 0 0 0 0 0 | Fail |
| T3 | 0 0 1 0 0 0 0 0 | Fail | 1 1 1 0 0 0 0 0 | Fail |
| T4 | 0 0 0 1 0 0 0 0 | Pass | 1 1 1 0 1 1 1 1 | Fail |
| T5 | 0 0 0 0 1 0 0 0 | Fail |  |  |  |
| T6 | 0 0 0 0 0 1 0 0 | Fail |  |  |  |
| T7 | 0 0 0 0 0 0 1 0 | Pass |  |  |
| T8 | 0 0 0 0 0 0 0 1 | Fail |  |  |
| MFS |  | (- , -, -, 0, - ,-, -, -) | |  | (-, -, -, 0, -, -, -, -) | |

Note that in this Table, OFOT generates eight test cases --T1 to T8, each of which mutates one factor of the original test case T0, and obtains the MFS(-, -, -, 0, -, -, -, -) because that only when the fourth factor is mutated, the generated test case will pass.

For FIC\_BS, it only needs four test cases—T1’ to T4’, and then determines the fourth factor is failure-inducing factor. The overall theory of FIC\_BS, in short, is: 1) when the last generated test case fails, the next generated test case should increase the factors to be mutated, 2) when the last generated test case passes, the next generated test case should decrease the factors to be mutated. This is because, when last test case fails, it means that test case still contains some MFS, so we need to mutate more parts to break the MFS, such that the failure-inducing factors can be revealed; and when the last test case passes, it means that we have already broken some failure-inducing factors, the next step is to narrow down the mutated parts, this because those mutated parts may not all be the failure-inducing factors, and we should filter those non-failure-inducing ones to obtain the MFS.

In this example, with T0 fails, FIC\_BS increases the mutated parts, so it chooses the left half of factors in T0, and changed it to be 1. The result of Pass indicates that T1’ breaks some failure-inducing factors of T0, and these failure-inducing factors must be contained in the left half. As T1’ passes, it next narrows down the scope of mutated parts, so it then breaks the left 4 factors of T1’ into halves (each half now contains two factors), then only keeps the left half (first and second factors) remaining mutated, i.e., value 1, and let other two factors back to un-mutated, i.e., value 0. The fail of T2’ indicates that T2’ does not break the failure-inducing factors. Based on this, we can get that the failure-inducing factors must lie on the right half (the 3rd and the 4th factors). With T2’ fails, it should increases the mutated parts, so it breaks down the two factors into halves, with 3rd factor to be the left half and the 4th factor the right half. It next let the left part, i.e., 3rd factor to be mutated to generate T3’, of which the result is fail. It still means that T3’ does not break failure-inducing factors. Hence, the failure-inducing factors can only lie on the right half, i.e., the 4th factor. Up to now, we can determine that the 4th factor must be one failure-inducing factor of the MFS. Next, it keeps the 4th factor to be the same as the T0, and mutated all the other factors to see if there is any other failure-inducing factors. The fail of T4’ indicates that it does not break any more MFS, hence, there is no other failure-inducing factors and the MFS is (-, -, -, 0, -, -, -, -). Note that, if not so, FIC\_BS will repeat the binary search process to get the remaining failure-inducing factors.

Considering that to completely explain how FIC\_BS works must consume much content of the paper, which will result in our paper hard to read (the main point of our paper, i.e., handling multiple failures, will be obscured), we decide to use OFOT again instead of FIC\_BS, to depict our approach in this paper.

Comment 2:

Second, I don't understand the sentence "Note that these mutated parts do not necessarily  
have to be value (0,0,-,-)" . To me if the fixed part is (-,-,1,1), and the original test is  
(1,1,1,1), the only possible mutated part is (0,0,-,-) otherwise you change the fixed part.

Response: Yes, if each parameters of the SUT has only two values, the only possible mutated part is (0, 0, -, -). However, if the number of values is more than 2, e.g., each parameter can be assigned to be 0, 1 or 2, then the mutated part can be (2, 2, -, -), (2, 1, -, -), (1, 2, -, -,) or (1, 1, -, -). According to this comment, we have emphasized this point to avoid ambiguity.

Comment 3:  
  
Third, the notion of "safe value" is not clear:  
- in "assumption 4" it is defined as "a value that is not part of any MFS".  
Should this mean that you know in advance what are the MFS? I thought that the  
goal was to find these.  
- at page 24, you give another definition "each parameter has one value that is not  
the part of any MFS". This is more understandable than Assumption 4 and should appear  
earlier in the text. Nevertheless, it does not explain how you can state that e.g.  
(-,-,-,1) is safe value when you don't know the MFS.

Response: Yes, our ultimate goal is to find all the MFS, and surely they are not known in advance. As a result, in theory, we cannot obtain these safe values before we identify all the MFS. Also, we agree that the sentence, i.e., "each parameter has one value that is not the part of any MFS", is more understandable than the original Assumption 4, and hence we replace the original expression with this one.

According to this comment, we realize that the Assumption 4 is not expressed properly, and only with that assumption we can still not obtain those safe values. In fact, this assumption should be presented to be “each parameter has safe value, and these safe values are all known in advance”, as what is firstly introduced in [1] and later in [2]. Also as said in [1] [2], this assumption, although reduces much number of test cases to be generated in MFS identification, itself is a very strong assumption. Hence, we need to discuss the negative effects of this assumption on MFS identification and possible alleviations.

In practice, we do not know the safe value in advance without any special information. Under such circumstance, what we can do is to use some different values other than the original value to mutate the original failing test case. There exists some pragmatic approaches to select the “safer” values. For example, we can select those values that have the least suspiciousness with faults, that is, the parameter values which rarely appear in the failing test cases. However, it cannot ensure that we can always get a safe value. When we happen to select an “unsafe” value, i.e., a value of some unknown MFS, it will result in an inaccurate MFS identification. For example, assume we need to identify the MFS in failing test case (0, 0, 0, 0), of which we assume the MFS should be (0, -, 0, -). Considering our identification use OFOT is as following:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| T1 | 1 | 0 | 0 | 0 | Pass |
| T2 | 0 | 1 | 0 | 0 | Fail |
| T3 | 0 | 0 | **1** | 0 | Fail |
| T4 | 0 | 0 | 0 | 1 | Fail |

Note that, T3 should have passed as we break the original MFS (0, -, 0, -). However, as we select an unsafe value (- ,- ,1, -), which itself is a MFS, we cannot get the accurate MFS (0, -, 0, -). As a result, the schema we identified will be (0, -, - ,-), which should be the sub-schema of the actual MFS. We have made some initial attempts to alleviate this problem [3] [4], e.g., to increase the number of test cases when we start one mutation. For example, we can generate test cases (0, 0, 1, 0), (0, 0, 2, 0) and (0, 0, 3, 0) instead of just one test case when we mutate the 3rd parameter of the original failing test case. This will increase the chance to obtain a correct MFS. Nevertheless, we think it still deserves further studies on the characteristics and possible alleviations of the safe assumption, as well as other assumptions in MFS identification, e.g., the assumption that needs to know the degrees of the MFS at first [5]. According to this comment, we have added these discussions of safe value assumption in the new version of the paper. (See page. )

[1] Martínez C, Moura L, Panario D, et al. Locating errors using ELAs, covering arrays, and adaptive testing algorithms[J]. SIAM Journal on Discrete Mathematics, 2009, 23(4): 1776-1799.

[2] Zhang Z, Zhang J. Characterizing failure-causing parameter interactions by adaptive testing[C]//Proceedings of the 2011 International Symposium on Software Testing and Analysis. ACM, 2011: 331-341.

[3] Nie C, Leung H. The minimal failure-causing schema of combinatorial testing[J]. ACM Transactions on Software Engineering and Methodology (TOSEM), 2011, 20(4): 15.

[4] Niu X, Nie C, Lei Y, et al. Identifying failure-inducing combinations using tuple relationship[C]//Software Testing, Verification and Validation Workshops (ICSTW), 2013 IEEE Sixth International Conference on. IEEE, 2013: 271-280.

[5] L. Ghandehari, Y. Lei, T. Xie, R. Kuhn, R. Kacker, Identifying failure-inducing combinations in a combinatorial test set, Proceedings of 5th IEEE International Conference on Software Testing, Verification and Validation (ICST), 2012.

Comment 4:

Fourth, although I understood corollary 4.3, I don't understand Corollary 4.4. Actually, I  
don't understand the example. The text of 4.3 and 4.4 is the same, you just replace  
passing by failing. In the example, you start from (0,0,0,0) and the new test is (0,0,0,1)  
so the fixed part is (0,0,0,-) and the mutated part is (-,-,-,1) (not -,-,-,0 as stated  
in the paper). So something must be wrongly expressed in corollary 4.4.

Response: Sorry to express wrongly in Corollary 4.4. As you said, the original Corollary 4.4 just replaced passing by failing of Corollary 4.3, and this is surely not correct. This error also negatively affects the understanding of Table X as said in the Comment 5. The cause of this error is that we misused the “mutated part” in this corollary. To explain this more clearly, we put the original two corollaries here to compare them.

**COROLLARY 4.3. *∀t ∈ Tunknown, if ∃t′ ∈ TP\_ known, such that t is obtained by mutating some parameter values of t′, and the mutated part are all safe values, then t must be a passing test case.***

**COROLLARY 4.4 (Original). *∀t ∈ Tunknown, if ∃t′ ∈ TF\_known, such that t is obtained by mutating some parameter values of t′, and the mutated part are all safe values, then t must be a failing test case.***

For COROLLARY 4.3, as the mutated part of *t* are all safe values, so it will not introduce any MFS. Combining the fact that *t* is mutated from a passing test case *t’* (indicating that *t’* does not contain any MFS), so the newly generated test case *t* also does not contain any MFS and will pass.

However, COROLLARY 4.4 is not correct as what it expressed originally. For an counterexample, if *t’* (0, 0, 0, 0) is a failing test case, and (- ,- ,0, 0) is the MFS, and we let (-, -, -, 1) be the safe value. Hence *t* (0, 0, 0, 1) is a passing test case as it does not contain the MFS (-, -, 0, 0). Note that here (- ,- ,-, 1) is the mutated part of *t*.

In fact, we had intended to express COROLLARY 4.4 as the following:

**COROLLARY 4.4 (New). *∀t ∈ Tunknown, if ∃t′ ∈ TF\_known, such that t’ is obtained by mutating some parameter values of t, and the mutated part are all safe values, then t must be a failing test case.***

Note that, the new COROLLARY 4.4 not only changes the passing by failing, it also changes the “mutated subject”. In the original COROLLARY 4.4, *t* is mutated from *t’* as the same as COROLLARY 4.3, while the new one let *t’* mutated from *t*. The new COROLLARY 4.4 can be proved as followed.

Proved: Assume *t* is a passing test case, as *t’* is mutated from *t*, and the mutated part are all safe values, hence according to COROLLARY 4.3, *t’* must be a passing test case, which is contradiction. Hence, *t* must be a failing test case, and COROLLARY 4.4 holds.

With the new COROLLARY 4.4, the example we showed in the original paper is easy to understand. In that example, we start from (0, 0, 0, 0) is known to be a failing test case. Note that here (0, 0, 0, 0) is *t’*, and it is mutated from test case *t* (0, 0, 0, 1). Here the mutated part of *t’* (0, 0, 0, 0) is (- ,- ,-, 0), which is safe value. Then test case *t* (0, 0, 0, 1) must be a failing test case. As suggested, we have rephrased this example to make it clearer.

Comment 5:  
  
Finally, I have problems with Table X.  
First, I don't know how you assume the safe values (see my previous remarks).  
Then, I understand how t2, t3 are infered from t4.  
I also understand that t1 is infered from t5.  
But I have difficulties to apply corollary 4.4 to obtain t14 from t16.  
But also, why did not you infer t15 from t16 (as you did for t14) using safe value (-,-,-,0)?

Response: As mentioned in the response of Comment 1, we have changed the approach description by using OFOT, and consequently all the following tables are rollback to the first version we have submitted. So, in order to response to this comment, we decide to post the original Table here for convenience.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ID | Test Case | | | | Outcome | State |
| T1 | 0 | 0 | 0 | 0 | pass | infer |
| T2 | 0 | 0 | 0 | 1 | pass | infer |
| T3 | 0 | 0 | ` | 0 | pass | infer |
| T4 | 0 | 0 | 1 | 1 | pass | known |
| T5 | 0 | 1 | 0 | 0 | pass | known |
| T6 | 0 | 1 | 0 | 1 | fail | known |
| T7 | 0 | 1 | 1 | 0 | - | unkonwn |
| T8 | 0 | 1 | 1 | 1 | fail | known |
| T9 | 1 | 0 | 0 | 0 | - | unkonwn |
| T10 | 1 | 0 | 0 | 1 | - | unkonwn |
| T11 | 1 | 0 | 1 | 0 | - | unkonwn |
| T12 | 1 | 0 | 1 | 1 | - | unkonwn |
| T13 | 1 | 1 | 0 | 0 | - | unkonwn |
| T14 | 1 | 1 | 0 | 1 | fail | infer |
| T15 | 1 | 1 | 1 | 0 | - | unkonwn |
| T16 | 1 | 1 | 1 | 1 | fail | known |

First, as we explained in comment 3, these safe values are assumed to know in advance. That is, we assume we have already known (0, -, -, -),(-, 0, -, -) ,(-, -, 0, -) and (-, -, -, 0) are safe values. Second, we made a mistake here, i.e., T14 (1, 1, 0, 1) cannot be inferred by T16 (Or T15 can also be inferred by T16 as mentioned in this comment). Instead, T14 should be inferred by T6 (0, 1, 0, 1) which is known to fail. According to the newly expressed corollary 4.4 (The original one leads the wrong inference, and causes the mistake), as T6 is mutated by T14 by changed the first factor of T14 to a safe value, and T6 itself is a failing test case, hence, T14 should be a failing test case.

Comment 6:

Finally, since I don't understand the details of FIC\_BS, I am unable to understand page 15  
in details, especially I don't follow the inferences of Fig 2 and 3.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ID | Test case | | | | Outcome | State |
| t1 | 0 | 0 | 0 | 0 | Pass | Infer |
| t2 | 0 | 0 | 0 | 1 | Pass | Infer |
| t3 | 0 | 0 | 1 | 0 | Pass | Infer |
| t4(t1’) | 0 | 0 | 1 | 1 | Pass | Known |
| t5(t3’) | 0 | 1 | 0 | 0 | F2 (F1) | Strategy |
| t6 | 0 | 1 | 0 | 1 | F1 | Infer |
| t7 | 0 | 1 | 1 | 0 | F1 | Infer |
| t8(t2’) | 0 | 1 | 1 | 1 | F1 | Known |
| t9 | 1 | 0 | 0 | 0 | - | Unknown |
| t10 | 1 | 0 | 0 | 1 | - | Unknown |
| t11 | 1 | 0 | 1 | 0 | - | Unknown |
| t12 | 1 | 0 | 1 | 1 | - | Unknown |
| t13 | 1 | 1 | 0 | 0 | F1 | Infer |
| t14 | 1 | 1 | 0 | 1 | F1 | Infer |
| t15 | 1 | 1 | 1 | 0 | F1 | Infer |
| t16(t) | 1 | 1 | 1 | 1 | F1 | Known |

Response: According to Comment 1, we have removed the description of FIC\_BS, and replaced it with OFOT. Consequently, all these figures are changed. We have improved the description of these figures, and what’s more, the re-expressed Corollary 4.4 make those inferences in a correct way. With respect to the original figures, we put Fig 2 here, and explain it, as well as those inferences. The original Fig 3 can be stated in the same way.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ID | Test case | | | | Outcome |
| t | 1 | 1 | 1 | 1 | F1 |
| t1’ | 0 | 0 | 1 | 1 | Pass |
| t2‘ | 0 | 1 | 1 | 1 | F1 |
| t3’ | 0 | 1 | 0 | 0 | F2 |

|  |
| --- |
| MFS of F1 |
| (-, 1, -, - ) |

.

The left table show the test cases FIC\_BS generated to obtain the MFS from failing test case (1, 1, 1, 1). As we have explained the process of FIC\_BS in Comment 1, the overall process can be regarded as a binary search mutated version of OFOT, to some extent. This process is an increment way, i.e., the test case generation depends on the result of the last test case. The completed explanation of this process can be obtained as said in Comment 1. Next, we select two test cases, i.e.,T3, and T6, as representatives, to explain how the inferences work:

As T3 (0, 0, 1, 0) can be obtained by mutated the 4th factor of T4 (0, 0, 1, 1) to be a safe value, and T4 is a passing test case, so T3 should be passing according to Corollary 4.3.

As T5 (0, 1, 0, 0) is a failing test case, and is mutated by the 4th factor of T6 (0, 1, 0, 1) to be a safe value, so T6 is a failing test case according to Corollary 4.4.

Detailed comments:

page 6 : "the THE specific fault information"

Response: Fixed as suggested.

page 7 : "which we will discussED later"

Response: Fixed as suggested.

page 11 : "focus on describing the FIC\_BS apprAOch"

Response: Fixed as suggested.

page 15, Fig 2 MFS of F1 should be (-,1,-,-) instead of (-,1,-,)  (- is missing)

Response: Fixed as suggested.

page 25, table XVII, versions Config space of Grep 2.6.3 : 2^5 \* 3^1 \* ??4^1?? actually, none of the options of grep 2.6.3 take 4 values.

Response: Fixed to be “2^6 \* 3^1”.

page 32 : "Although our approach needS more " (s missing)

Response: Fixed as suggested.

page 33 : "FDA-CIT... approach handleS masking" (s missing)

Response: Fixed as suggested.

page 38, section related works : you refer to OFOT and SOFOT, which were covered in the previous  
version but in the current version, the reader don't know these at this point.

Response: Fixed. We have re-introduced the OFOT in the new version.

Referee: 2  
  
Comment 1.

Originally, I was having the concern that the advantage over simpler approaches is limited. The authors responded that "This number is about 7.1% to 14.2% of the overall number of test cases needed for MFS identification." I still think such an improvement is not substantial. The technique is for in-house testing and debugging. Such savings may not mean much.

Response: We agree. According to this comment and Comment 5, we have removed the ILP-based approach, and only keep the approach with simple random replacement strategy. We also added more real subjects according to Comment 2. As a result, we obtain some new observations: For subjects with large configure space, the random-based approach can perform far better than the simplest “distinguishing failures” approach, with which the advantage can reach to .

Comment 2.

Evaluation on three projects is still quite thin for a TOSEM paper.

Response: According to this comment, we have added 4 more real subjects (Log 4j, Joda-time, jsoup and Apache-Commons-cli), in which Log 4j is a java log tool, Joda-time is is the de facto standard date and time library, jsoup is Java HTML Parser, Apache-Commons-cli is a command line parser. Hence, we now have 7 real projects with versions. Additionally, we have remove all those synthetic ones. The total subject as well as subjects are as following table.

Additionally, we have post them in the website: . That can reproduce those errors.

joda-time 2.8.2 Paser error For Long Max and Min <https://github.com/JodaOrg/joda-time/issues/297>

Interval.parse().toDurationMillis() returns wrong values <https://github.com/JodaOrg/joda-time/issues/296>

Joda-time 2.9.2 Getting ArrayIndexOutOfBoundsException exception <https://github.com/JodaOrg/joda-time/issues/361>

Etc/GMT not recognised as timezone id in 2.9.1 <https://github.com/JodaOrg/joda-time/issues/347>

Joda-time 2.3 addDays(0) changes value of MutableDateTime <https://github.com/JodaOrg/joda-time/issues/77>

none standard PeriodType without year throws exception <https://github.com/JodaOrg/joda-time/issues/79>

jsoup 1.9.1 infinite Fix getting stuck on non-ascii tags <https://github.com/jhy/jsoup/issues/704>

Bug in Element.insertChildren() <https://github.com/jhy/jsoup/issues/689>

jsoup 1.9.2 JSoup Created XML Document Contains Invalid XML Declaration String <https://github.com/jhy/jsoup/issues/652>

Parse failed with org.jsoup.select.Selector$SelectorParseException when selector has unbalanced <https://github.com/jhy/jsoup/issues/611>

Common-cli 1.3.1 Optional argument picking up next regular option as its argument <https://issues.apache.org/jira/browse/CLI-265?jql=project%20%3D%20CLI>

Short/Long options can be called using either '-' or '--' hyphens with no limitation <https://issues.apache.org/jira/browse/CLI-264?jql=project%20%3D%20CLI>

Common-cli 1.2 MissingOptionException.getMissingOptions() returns OptionGroup, not just String <https://issues.apache.org/jira/browse/CLI-220?jql=project%20%3D%20CLI>

Quoted string parsing <https://issues.apache.org/jira/browse/CLI-213?jql=project%20%3D%20CLI>

Commons Lang 3.4 org.apache.commons.lang3.math.NumberUtils.isNumber bug <https://issues.apache.org/jira/browse/LANG-1216?jql=project%20%3D%20LANG>

NumberUtils.createNumber() method lost precision sometimes <https://issues.apache.org/jira/browse/LANG-1215?jql=project%20%3D%20LANG>

3.3.2 NumberUtils#createNumber() returns positive BigDecimal when negative Float is expected <https://issues.apache.org/jira/browse/LANG-1087?jql=project%20%3D%20LANG>

DiffBuilder.append(String, Object left, Object right) does not do a left.equals(right) check <https://issues.apache.org/jira/browse/LANG-1081?jql=project%20%3D%20LANG>

Comment 5.

The argument of using ILP is not that convincing. I still think simple search is good enough. Maybe some empirical study would be helpful.

Response: We agree. According to this comment, we have removed the ILP-based approach, and only keep the one using random replacement. Also as suggested, we also added more empirical studies, and obtained some more observations.

Comment 6.

There are still quite a number of typos and grammatical errors. To name a few:

Response:   
  
Page 23, "must aware"

Response: Revised to be “must be aware”.  
  
Page 23, "the MFS identified are"

Response:   
  
Page 17, what is "failures of other failures"? This term is being used in multiple places.

Response:   
  
Page 17, "(1)to", "(2)to" => "(1) to", "(2) to"

Response: Fixed as suggested

page 16, "satisfied test case" => "satisfying test case". It happens in many places.

Response:   
  
page 4, " an theoretical framework"  
  
Response: Revised to be “a theoretical framework".