Undergo a Major Revision. 14-Feb-2018.  
1. “Summary of Changes”. 2. Response to Comments. 3. Identify a teaser image 4. Revision.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
Editor Comments  
  
The referee believes that some of the contributions are not adequately supported by the empirical evidence, and therefore questions whether the paper overall has enough of a contribution to form the basis of a publication in the journal.  
  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
  
The changes made in the revised manuscript are highlighted in Red in the “summary of changes” according to comments of Reviewer 1, and highlighted in Blue according to comments of Reviewer 3.   
  
Reviewer: 1

Comment 1:

However, can the authors clarify whether and why the feedback checking mechanism is something inherent to their suggested approach, or rather orthogonal that can be embedded into SCT and FDA-CIT as well? Especially considering the conclusion for Q5 that ICT handles better the cases where non-determinism and non-safe values exist thanks to this feedback checking process.

**Response: As suggested, we have added one paragraph in Section 4.2.1 (Page 9, the red part) to clarify this. Our main point is that this feedback checking mechanism can also be embedded into SCT. However, there are two drawbacks that can limit the improvement of this mechanism. First, the fix of wrongly identified MFS cannot be used in other test cases. Second, it costs SCT more to embed feedback checking mechanism.**

**With respect to FDA-CIT, our feedback checking mechanism cannot be directly applied on it. This is mainly because the MFS identification process of FDA-CIT is much different from that of ours, which is a post-analysis classification tree method. It does not need any additional test cases, while our approach needs additional test cases to refine the accurateness of MFS identification. Another problem to apply feedback checking mechanism on FDA-CIT is that it may cause the over-fitting issues of the classification tree method of FDA-CIT.**

Comment 2:

I feel that the statement "Traditional covering array usually offer an inadequate testing due to Masking effects" (3.3) needs to be refined. This is true only when considering a single execution of the test set. In practice, it is often the case that the set is rerun until all test cases pass. A passing regression is even a commonly defined gating condition in Cloud development. [5] mentions fix and rerun as a viable option to handle masking effects, though of course it has its limitations - it would be better to progress with testing and cover as much as possible while waiting for the fixes to be available, which is what ICT and FDA-CIT offer.

**Response: We agree. We have refined this sentence according to your comments. The changes can be found in Section 3.3 (red part of paragraph 1 and 2). Specifically, we have emphasized the point that “Running traditional covering array at one time is inadequate, and testers usually try to repeatedly run the covering array in practice to achieve a relatively adequate testing”.**

Comment 3:

Finally, I do not understand the reasoning for the choice on how to handle constraints. The authors indicate they choose the dynamic method since "it can directly \*be\* applied into our framework". However it seems to me that both methods can be applied to the framework. If you already have a mechanism for handling forbidden tuples for previously discovered MFS, does it really matter if you provide forbidden tuples during initialization or on-the-fly? Where it seems to matter is in the high performance penalty you would pay by using the dynamic method to discover the minimal forbidden tuples one by one through invalid test generation, when in typical real-world models hundreds or even thousands of such tuples exist (in many cases more than the number of failures). In the static approach, the user needs to pre-define constraints, but usually these are known, and each constraint typically concisely summarizes a large set of forbidden tuples. You could also use a combined approach, where the user specifies known constraints even if they are incomplete in order to reduce their appearance during execution and achieve performance savings.

**Response: It is a very good question, and we agree with you that both methods can be applied to the framework. Your suggestion that combining both static and dynamic ways of handling constraints is also very useful. We believe that, in practice, it is wise to first give initial settings of constraints (especially when there are too many constraints) such that the generation approach can efficiently avoid invalid test cases, otherwise we need to identify these constraints one by one. The dynamic way of identifying constraints and then avoiding them is also useful, because we may not always get an accurate input model at the first time and some constraints may be ignored at some time.**

**According to your comments, we have rephrased the sentences for describing why we chose the dynamic way of handling constraints in our empirical studies (See Section 4.2.2, paragraph 1, the red part). Our main point is that there are two reasons for this choice. First, there are not many constraints in our empirical study such that the dynamic way of identifying them and forbidding them will not affect the efficiency too much. Second, the dynamic process of handling constraints is similar to the way that we identify the MFS, so our framework does not need to be modified a lot for handling constraints.**

Reviewer: 2  
  
Public Comments (these will be made available to the author)  
The authors addressed my comments well and I am happy to suggest the acceptance of the paper.

**Thanks a lot**!  
  
Reviewer: 3  
  
  
Comment 1:

One contribution they claim to have is the feedback checking mechanism given in Algorithm 2, which after identifying an MFS, tests a predefined number of test cases to check to see if a condition containing MFS really fails or not. When they are choosing these additional test cases they try to pick the ones that are the most different than the original failing test case, which seems to be an idea already studied in another paper. However, in the experiments no data is provided to show how much the proposed approach gained from this functionality. For example, what is the performance of the proposed approach with and without this functionality?? Therefore, it is not clear how valuable this contribution is.

**Response: We agree. According to this comment, we conducted an experiment to evaluate the approach without this feedback checking mechanism. Specifically, we created a mutation version of ict by removing the feedback checking mechanism from the original ict approach, and then applied this approach on testing the software subjects and identifying the MFS. The results of our experiment show that, without feed-back checking mechanism, the number of test cases generated by ict reduced, but the quality of MFS identification and tested-t-way coverage decreased significantly. It indicates that the additional test cases generated in feedback checking mechanism is worthwhile, and it is beneficial to adopt the feedback checking mechanism in the CT process in order to obtain a better MFS identification result and a higher tested-t-way coverage.**

Comment 2:

What was the point having the deterministic failure in the experiments. Didn’t it greatly increase the F-measures reported in the paper. Even so, the F-measures were significantly dropped and as noted in the paper between the non-determinism levels of 0.4 and 0.8 (where the failures manifested themselves between 40% and 80% of the time) it was about and below 0.4. Therefore, it doesn’t really make sense the performance of ICT (the proposed approach) to other approaches because an F-measure of 0.4 is not acceptable. When this coupled with the fact that had more non-deterministic MFSes were used, the F-measures would have been even smaller, the statement (page 20, 2nd column, lines 55-58) “this result indicates that act can handle the non-deterministic failures properly” becomes simply not acceptable.

**Response: We agree with your comments. According to this comment, we remove the deterministic failure in our experiment, and only focus on the non-deterministic failure. As expected, the results (f-measure of MFS identification) of all the three approaches decreased significantly. Hence, we agree to your comment that “this result indicates that ict can handle the non-deterministic failures properly” is not appropriate, and remove this statement. Besides this, we consider a new strategy to alleviate this non-deterministic issue. We call this strategy “redundancy of test case execution”, i.e., we repeatedly run one test case to check whether it fails or not instead of just one time. We conducted one additional experiment (Section 5.8.2, paragraph 5 and 6) to evaluate the performance of this strategy and found that there was a significant improvement on the quality of MFS identification. It indicates that the redundancy of test case execution is one potential approach to handle the non-deterministic failures problem.**

Comment 3:

Another claimed contribution is the experiments which were performed to evaluate ICT on models with no safe-values. First, these experiments were also carried out using a quite questionable set of synthetic data where for each value of a parameter was made to appear in an MFS. However, no information was provided about the cardinality of these manually generated MFSes. How many unique parameters each had?? This is important because if they had large cardinalities they were highly unlikely to be hit during testing. Another question is how many times these MFSes actually caused failures during the MFS identification times?

**Response: Yes, we agree with your comment about “if they had large cardinalities they were highly unlikely to be hit during testing.” Hence, we re-modeled the** **cardinalities of these non-safe MFSes, and let them be more easily to be triggered. Now, we have provided these information of MFS, as well as the inputs model, in Table 28 (Section 5.8.1). As suggested, we also provided the number of times these MFSes actually caused failures during the MFS identification times in Figure 17 and 18 (Section 5.8.3). We can observe that in our experiments, these non-safe MFS were triggered frequently (Specifically, for each time of MFS identification, these non-safe MFS were triggered by about 22 times at average for ict, and 7.3 times for sct).**

Comment 4:

So, multiple faults (defects) are not actually handled by the proposed approach. That is, the multiple MFSes used in the experiments were assumed to be caused by exactly the same defect. So, the system under test is assumed to have exactly one defect, which is indeed also evident in Table 7. First, this significantly hinders the applicability of the proposed approach as this is a quite strong of an assumption to have for a testing approach. Second, the effect of multiple faults/defects should have been studied. Third, the comparisons made against FDA-CIT is not fair as FDA-CIT seems to design to work with not only multiple defects but also with multiple defect-test case pairs.

**Response: As suggested by the reviewer, we have added the experiments with multiple defeats (See Section 5.6, blue part). Specifically, we considered five different software subjects with multiple defeats, and then conducted the three approaches, i.e., ict, sct, and fda-cit, on testing them and identifying the MFS. We found that most results matched pretty well with the results obtained from the experiments of single defeat. Particularly, ict performed better at MFS identification and reducing of number of test cases that containing multiple MFS, and fda-cit generated the least number of test cases and covered the most number of tested-t-way schemas. When compared with ict and fda-cit, approach sct was normally in between.**

Comment 5:

I change the example: we have one defect causing multiple  
MFSes:  
abs  
000  
100  
But this time a is a ternary option and b ,c, and d are binary options. Since not all the values of a parameter will be exercised by ICT, starting with (0, 0, 0, 0), we would have  
abcd  
0000 F  
1000 F  
0100 P  
0010 P  
0001 F  
(correct me if I am wrong) which will mark (b=0, and c=0) as an MFS and it would be wrong! Note that this is different than the discussion about why finding the minimum MFS is expensive. This is simply the case where the algorithm wrongly identifies a combination as MFS. Therefore, the authors, I guess, could have come up something with an “approximate MFS - AMFS” definition and rather than stating that ICT finds MFS in many places in the text, which is definitely confusing and not seem to be right, as shown by the counter example above, could have stated that ICT finds AMFS.

**Response: This example you provided is not the way how ICT actually works. We believe the misunderstanding of ICT is because of we did not give more details of our algorithms (as you mentioned in Comment 6), and hence, we offer more details of ict, as well as sct, on the Appendix as suggested by Comment 6. Additionally, we have offered an example which is similar to what you provided in this comment in Section 4.4 (blue part), which shows how ict works on such conditions. With respect to this specific example, ict will work as the following Table. The test cases T1-T5 are the same as you provided in this comment, but ict will not stop in such condition, because it needs to check this schema. We can learn that the checking process (T6) let ict be aware of that (-, 0, 0, -) is not the real MFS (because T6 contains (-, 0, 0, -), but passes during testing), and then re-identify the MFS until it passes the validation.**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Generation** | | | **Identification** | | |
| **T1** | **0 0 0 0** | **Fail** |  |  |  |
|  |  |  | **T2** | **1 0 0 0** | **Fail** |
|  |  |  | **T3** | **0 1 0 0** | **Pass** |
|  |  |  | **T4** | **0 0 1 0** | **Pass** |
|  |  |  | **T5** | **0 0 0 1** | **Fail** |
|  |  |  | **Candidate MFS (-, 0, 0, -)** | | |
|  |  |  | **Checking** | | |
|  |  |  | **T6** | **2 0 0 1** | **Pass** |
|  |  |  | **Re-identify** | | |
|  |  |  | **T7** | **2 0 0 0** | **Pass** |
|  |  |  | **T8** | **0 1 0 0** | **Pass** |
|  |  |  | **T9** | **0 0 1 0** | **Pass** |
|  |  |  | **T10** | **0 0 0 1** |  |
|  |  |  | **Candidate MFS (0, 0, 0, -)** | | |
|  |  |  | **Checking** | | |
|  |  |  | **T11** | **(0, 0, 0, 1)** | **Fail** |
|  |  |  | **MFS (0, 0, 0, -)** | | |

**Nevertheless, we agree with your comment that there is chances that algorithm wrongly identifies a combination as the real MFS.**

Comment 6:

Part of the confusion here is that although two algorithms were given regarding certain functionalities in ICT the main algorithm was not actually given, so it is not possible how exactly ICT proceeds.

**Response: As suggested by this comment and the latter comment (Comment 14), we appended the whole algorithm of ICT (including all the functionalities) in the Appendix. The detail of the functionalities in each algorithm was also described along with each algorithm.**

Comment 7:

Regarding the response to Reviewer 3’s comment #16: Authors states that “we explicitly show the assumption used in our study (See Section 2.2).” However, in Section 2.2. the only approach, the assumptions of which are explicitly states, is the OFOT approach. They don’t actually talk about the assumptions of ICT. Because for example Assumption #2 states basically states that no multiple MFS, but ICT (or at least the rest of the paper) is all about multiple MFSes. Furthermore, the single defect assumption we have discussed above is not discussed anywhere in the paper.

**Response：At first, we need to correct your statement about Assumption 2. Assumption 2 is not about “no multiple MFSes”, but is about “safe value”. In fact, OFOT can work on SUT with multiple MFSes, as long as these multiple MFSes will not be introduced when OFOT focus on identifying one specific MFS. Besides, when multiple MFSes are contained in one test case, OFOT algorithm fails in identifying any of them, but we have already discussed that ICT can reduce the appearance of test cases that contain multiple MFS (Section 4.3). Furthermore, as you suggested in the next comment (Comment 8), we have shown the performance of ICT and SCT on test cases with multiple MFSes.**

**Hence, we guess what you actually mean in this comment is that we did not discuss how ICT handle the safe value assumption (Assumption 2). As suggested, we added one paragraph to discuss how ICT can alleviate the effects of safe value assumption (Section 4.3, 4th advantage of ICT, blue part), of which the main point is that the feedback checking mechanism make ict discard the wrongly identified schemas because of the safe value assumption, and hence, it gives ICT more chances to refine the test cases generated for MFS identification. Additionally, we also added one example in Section 4.4, as suggested in your latter comment (Comment 13), to show how ict refined the MFS identification result on the condition that the safe value assumption is not satisfied.** **With respect to multiple defects, as also suggested in Comment 4, we have conducted one more experiment with multiple defeats (See Section 5.6, blue part).**

Comment 8:

A related concern is that although ICT seems to be mostly about multiple MFSes, it does not seem to offer a mechanism to address this issue if a given failing test case has multiple MFSes. It simply attempts to avoid generating test cases with multiple MFSes by opportunistically hoping to hit test cases with single MFS before it hits test cases with multiple test cases, so that the single MFSes can be ruled out to reduce the chance of hitting multiple MFSes. Since there is no guarantee for this, show ICT performs on test cases with multiple MFSes is still a valid question? What would be the F-measure obtained for these cases?

**Response: According to this comment, we focused on the f-measure obtained by the proposed approach on the test cases that contain multiple MFS. Our results are added in Section 5.7.2 (the 6th paragraph). Our results mainly showed that approach ict outperformed sct on MFS identification on test cases that containing multiple MFS. Additionally, the condition that multiple MFS appear in one test case has large negative effects on sct, while has a relatively slight influence on ict. The reason why ict still outperformed sct is that, the feedback checking mechanism gives ict more tries to refine its MFS identification, and there is more chances that ict can obtain an accurate MFS identification result.**

Comment 9:

Regarding the response to Reviewer 3’s comment #8: Correct me if I am wrong, in the original submission, ICT used FIC in the MFS identification phase and the reason was that FIC performed better than OFOT. In this submission, ICT uses OFOT, because now it performs better. This could happen but I believe the authors should share their insights with the reader.

**Response: The reason why we used the OFOT instead of FIC is not because OFOT performs better than FIC. We used OFOT for ICT is to make a fair comparison according to your original comment (SCT and ICT should have implemented with the same MFS identification approach for a more reliable comparison). Actually, the reason why our new ict performed better than the original one is because we re-designed framework, i.e., adding the feedback checking mechanism to our framework. This mechanism significantly improved the quality of our MFS identification.**

Comment 10:

Section 5.6.2, paragraph 2: This paragraph basically indicates that if a failure caused by an MFS is not hit then that MFS is not included in the F-measures. Why not? Isn’t it a failure that the approach missed the MFS? Even if for some reason they need to be discarded, in the experiments statistics about the number of failures and number of unique failures found by each approach should be given. Because the number of MFSes hit, will definitely affect the F-measures computed, which should not have been the case. Because this would favor an approach that hits, for example, one MFS out of many and correctly localizes it over an approach that hits all the MFSes but failed to correctly localize all of them.

**Response: Sorry for the misunderstandings we may cause, but we did computed the MFS by considering the MFS that was not hit by the test cases generated in our paper of the previous version. And this is why the f-measure of our approaches did not reach 100% when the probabilities of non-deterministic failures is around 0.01 (is hard to hit). Under such condition, the precise is 1 (because we accurately identified another deterministic failure), and the recall is only 0.5 (only the deterministic failure is identified, while the non-deterministic MFS was not triggered and detected). The fact that we have included the missed MFS can also be proven in our new version of paper, in which when probabilities of non-deterministic failures was around 0.01, the f-measures of all the three approaches were 0 (because we have removed deterministic failure according to your Comment 2).**

Comment 11:

Furthermore, due to the all the assumptions made in the paper and the emphasis on doing this in a dynamic manner is much better than doing it in a static manner, the approach should have been compared to static Error Locating Arrays (ELAs). This should not be a problem because for the experiments carried out in this paper an ELA will be a standard (t+d)-way array, which will guarantee the identification of the MFSes up to and including the cardinality of d. So, the authors can generate standard covering arrays of appropriate strengths and compare their sizes to the numbers of test cases required by their approach.

**Response: As suggested by the reviewer, we have added one section (Section 5.9) to show the results of ELA. We obtained the same conclusion as [1]. That is, although ELA can identify the MFS accurately, it needs to generate much more test cases than the dynamic approaches. Additionally, it also needs to know the number and degree of the MFS at first, which limits its application in practice.**

**[1]C. Yilmaz, E. Dumlu, M. Cohen, and A. Porter, “Reducing masking effects in combinatorial interaction testing: A feedback driven adaptive approach,” Software Engineering, IEEE Transactions on, vol. 40, no. 1, pp. 43–66, Jan 2014.**

Comment 12:

There is a section on constraints in the paper (Section 4.2.2), which simply talks about the constraints identified by ICT. However, it is still not known how ICT picks values for the parameters if no values could be chosen due to the known and/or discovered constraints. For example, ICT wants to test a=0, but a=0 is invalided by some constraints. Then what???

**Response: According to this comment, we added more details (Section 4.2.2, 4th paragraph, blue part) to further clarify how the constraints are handled by our approach in this paper. Specifically, Our constraints handling techniques is based on the study [1]. There is two important part of this technique to handle the issue of this comment. The first part is updating uncovered schemas. That is, after one constraint or one MFS is obtained, we will update all the schemas that are still needed to be covered. This part is done by computing the compatibility [1] between the uncovered schemas with those known and discovered constraints. After this, all the possible implicated constraints (Not known prior, nor explicitly discovered), and hence, our algorithm will not be stuck in the unstoppable condition that some schemas cannot be covered. The second part is that, for one test case that is generated by our approach, we will compute the** **satisfiability of the value under selected for each parameter [1]. Specifically, for one pending value of one specific parameter, we will first use SAT solver to find if there is a solution (one possible test case) that contain this value and not violate any of these constraints (including implicated ones). If the solver returns true, which means we can find one satisfied test case, then this value can be selected as one candidate value for that parameter. Otherwise, this value will be discarded. Due to these two steps, ICT will not face the condition that we need to test a=0, but a=0 is invalided by some constraints, because this will result in a false result (non-valid test case can be obtained) by the solver.**

**[1] M. B. Cohen, M. B. Dwyer, and J. Shi, “Constructing interaction test suites for highly-configurable systems in the presence of constraints: A greedy approach,” Software Engineering, IEEE Transactions on, vol. 34, no. 5, pp. 633–650, 2008**

Comment 13:

It is stated in the paper (Section 4.4, paragraph 2) that if Algorithm 2 returns false (i.e., if one passing test case is found containing the previously identified MFS), the original failing test case is retested. How does this work? No detail is given in the paper.

**Response: Yes, the demo example only shows when the condition that Algorithm 2 returns true. As suggested, we added one additional example (5th and 6th paragraphs of Section 4.4, blue part) to show what our approach will do if Algorithm 2 returns false. We used a similar example you mentioned in Comment 5. Specifically, ICT wrongly identified the MFS in the failing test case (0, 0, 0, 0) at the first time, i.e., it identified the schema (-, 0, 0, -) to be the MFS instead of the real MFS (0, 0, 0, -). The additional checking test case (2, 0, 0, -) passed, indicating that (-, 0, 0, -) was not the real MFS. Hence, we re-identified the MFS in (0, 0, 0, 0) by additionally generated four test cases. These four test cases should be as different of what has been generated before as possible in order to cover more un-covered schemas. As shown in that example, ict correctly identified the real MFS (0, 0, 0, -) at the second time.**

Comment 14:

In Section 3.4. Table 5, how does the augmented SCT actually generate t14 and t19 given the failing test case t4 and previously identified MFS (-,0,-,-)?? No detail is given about this in the paper. Again, the algorithms for ICT and SCT should be given in the paper.

**Response: As suggested, we explicitly show these two algorithms for ICT and SCT in the Appendix.**

**With respect to the example in Table 5, the augmented** **SCT generated t14 and t19 to validate if there is any MFS other than the previously identified MFS (-,0,-,-). Specifically, test case t4 (1 0 0 1) contained previously identified MFS (-,0,-,-), and hence, we did not know whether it contained other MFS. The augmented** **SCT then generated t14 (1, 1, 0, 1) by mutating the value of the second parameter of test case t4 (1, 0, 0, 1) from 0 to 1 (note that it can be any value different from the original value 0 of t4), and as a result, it removed the previously identified MFS (-,0,-,-). Then the augmented** **SCT found t14 still failed, which indicated that t4 must contain other MFS which is different from (-, 0, -, -). Therefore, the augmented** **SCT needed to identify the different MFS in t14. The same as t14, the augmented** **SCT generated t19 to find if there is any MFS other than (-, 0, -, -) in test case t7 (2, 0, 1, 1). It was generated by was also generated by mutating the value of second parameter of test case t7 (2, 0, 1, 1) from 0 to 1. t19 passed after testing, and hence, the augmented** **SCT did not need to start the MFS identification. According to this comment, we added more detail (Section 3.4, 2rd paragraph, blue part) to describe how the augmented SCT actually generates t14 and t19.**

Comment 15:

Do the algorithms and the equations, at least the ones given in the paper, really enumerate and store all valid test cases (all possible combinations of parameter values) as they suggest they do, which obviously not scalable at all???

**Response: No, we do not really enumerate and store all valid test cases. Our eqution is to formally describe a unified framework of our approaches. In the implementation of ICT, we just simply use greedy approach to generate the test cases (select the parameter value that has the most un-covered schemas that contain it). This greedy method is the same as what AETG selects test case [1]. Additionally, for these conditions that the test case needs to satisfy, e.g., do not contain any constraints, we just use SAT solver to ensure it will not violate these restrictions [2].**

**[1] D. M. Cohen, S. R. Dalal, M. L. Fredman, and G. C. Patton, ”The aetg system: An approach to testing based on combinatorial design,” Software Engineering, IEEE Transactions on, vol. 23, no. 7,pp. 437–444, 1997**

**[2] M. B. Cohen, M. B. Dwyer, and J. Shi, “Constructing interaction test suites for highly-configurable systems in the presence of constraints: A greedy approach,” Software Engineering, IEEE Transactions on, vol. 34, no. 5, pp. 633–650, 2008**