Paper Title: Adaptive Partition Testing

Manuscript ID: TC-2018-02-0064

Dear Editor,

Thank you for your email on May 2, 2018 regarding our paper titled "Adaptive Partition Testing" submitted to IEEE Transactions on Computers (Manuscript ID: TC-2018-02-0064).

We are submitting a new version of the paper, in which we have made revisions to address and respond to each comment from the reviewers. Below are the detailed responses to the comments.

We look forward to hearing from you.

Yours sincerely,
Chang-ai Sun, Hepeng Dai, Huai Liu, Tsong Yueh Chen, and Kai-Yuan Cai

Response to comments of reviewers

In the following, unless otherwise specified, all comments refer to the revised version of the paper.

Reviewer 1's comments

***R1C1**: This paper considers the problem of combining partition testing (PT) and random testing (RT). It thus builds on random partition testing (RPT) and dynamic random testing (DRT) by introducing two new methods for updating probabilities. This leads to two new approaches called Markov-chain based APT (MAPT) and reward-punishment APT (RATP).*
Having introduced MAPT and RATP, the paper then describes experiments that compared these to RPT and DRT. The experiments were on three systems of moderate size, with there being several versions of each and three levels of granularity (in partitioning). This leads to 60 cases.
The results reported appear to be promising. The two new approaches outperformed RPT and DRT if one just considers the number of test cases required to find the first fault and also the additional test cases to find the second fault. The results were a little weaker if one also takes into account the time taken - since additional computation is required.
Overall, this is a potentially interesting paper. The problem considered is relevant, the paper is well written, and the results are largely promising.

Response: Thank you for the endorsement.

Action: None.

***R1C2**: … The results were a little weaker if one also takes into account the time taken - since additional computation is required …*

Response: Thanks for the comment. It is true that our approach incurs additional computation on updating probability profiles compared with traditional partition testing or random partition testing. However, such computation cost could be almost ignored in particular for programs whose executions take a long time. As shown in our experiments, the performance of RAPT is significantly better than that of RPT, while MAPT marginally outperforms RPT. In other words, such additional computation cost is compensated by the saving of fewer test executions.
Action: In the revised version, we added a short discussion on the computation cost in the end of Section 4.2.

***R1C3**: However, the current version appears to have a number of points that need resolving. These including the following: … In RAPT you use the natural log of Rew_i - it would be interesting to hear why/be provided with some intuition.*

Response: The basic idea of RAPT is to quicker select those potential fault-revealing test cases for execution via a reward and punishment mechanism. Accordingly, two parameters Rew_i and Pun_i are used to determine to what extent a partition c_i can be rewarded or published, respectively. The intuition behind such a mechanism is based on the well-known observation, that is, faults normally intend to cluster in some codes. This indicates that if a fault is detected by a test case in a partition c_i, more faults likely remain in the corresponding codes and thus the execution of more tests from c_i will be more productive in fault detection. In this context, Rew_i will be increased and Pun_i will become 0.

Furthermore, we use the nature log of Rew_i (i.e. ln Rew_i) for updating the probability of a partition c_i (Refer to Formula 10). The main reason is explained as follows. When a fault is detected in c_i, we would expect that more faults can still be detected in c_i. However, the situations may change after such an expectation is satisfied for several times. If the Rew_i is used, the probability increase of c_i is linear, which cannot differentiate the values of faults being detected in c_i at different stages. However, the use of ln Rew_i can overcome the above limitation, because the curve of ln Rew_i first rises steeply and then rises slowly. Keep this hypothesis in mind, we adopt ln Rew_i rather than Rew_i for implementing the probability update of c_i.

Action: In the revised version, we clearly explain the intuition behind the reward and punishment mechanism in the beginning of Section 2.3.

***R1C4*: There are complexity arguments for MAPT and RAPT that argue that the additional time taken in constant. However, these arguments assume that m (the number of partitions) is fixed. Is this reasonable? I would have expected m to be considered in any complexity argument. Maybe you could give the complexity in terms of m and then point to it being constant if m is fixed?**

Response: Sorry for this mistake. For a given program, m (the number of partitions) should be fixed as discussed in Section 3.4.1, while m is variable for different programs. We admit that in a general situation, the additional time taken by MAPT and RAPT should not be constant, although it should not be large. Furthermore, in practice, m is often much smaller than n, and the time complexity mainly depends on n.

Action: In the revised version, we have followed the reviewer's suggestion and changed the discussions on the time complexity in Section 2.2 and Section 2.3.

***R1C5*: Measures used. You correctly note that the F-measure is limited in this case - essentially, it tells us nothing about one of the mechanisms in your approaches (what happens if a failure is found). The F2-measure helps but it is still limited - it only allows one adjustment for finding a failure. Maybe you could consider another approach that gets around this limitation? Note that ideally we care about number of faults found not number of failures.**

Response: Sorry for the misunderstanding. ~~In the original version, we do mention the stop criteria in the algorithm for both MAPT and RAPT. That is, when a fault is~~

~~detected, the testing process continues only when the termination condition is not satisfied; otherwise, the testing process is stoped.~~ Accordingly, in this study, we consider two testing scenarios, namely when a fault is detected, the testing process is paused and debugging is started, and when a fault is detected, the testing process continues without a pause. For the former, F-measure is a suitable metrics, while for the latter, F2-mearsure appeals. Furthermore, F2-mearsure appeals for continuous adjustments rather than one adjustment since it measures the number of test cases required for detecting the first fault is detected in the context of the second fault being detected. It can be further generalized to measure the number of test cases required for detecting the $i+1^{th}$ fault is detected in the context of the i^{th} fault being detected in an iterative way.

Action: In the revised version, we have clearly stated what happens if a failure is found in Section 2.2 and Section 2.3, and added a discussion on appropriateness of measures used in Section 3.3.2.

***R1C6*: Experimental results. Here it is states that the Holm-Bonferroni method was used. However, my understanding is that the Holm-Bonferroni method simply adjusts p-values to take into account there being multiple statistical tests. It thus seems that you still need to apply a statistical test, using the adjusted p-value, and you say nothing about this.**

Response: We actually conducted hypothesis testing based on the Holm-Bonferroni method to determine which pair of testing techniques has significant difference in terms of each metric. Across the whole study, for each pair of testing techniques, denoted by technique *a* and technique *b*, the null hypothesis (H_0) was that *a* and *b* had the similar performance in terms of one metric. All the null hypotheses were ordered by their corresponding p-values, from lowest to largest; in other words, for null hypotheses H_0^1, H_0^2, …, we had $p_1 \leq p_2 \leq \ldots$. For the given confidence level $\alpha = 0.05$, we found the minimal index h such that $p_h > \alpha/(N+1-h)$ (where N is the total number of null hypotheses). Then, we rejected H_0^1, H_0^2, …, H_0^{h-1} that is, we regarded the pair of techniques involved in each of these hypotheses to have statistically significant difference in terms of a certain metric. On the other hand, we considered the pair of techniques involves in each of H_0^h, H_0^{h+1}, … not to have significant difference with respect to one metric, as these hypotheses were accepted.

Action: We have added the relevant content to explain the details of hypothesis testing more clearly in Section 4.1.

***R1C7*: The summary tables (e.g. Tables 6, 7) are unclear. It seems odd that when comparing methods M1 and M2, we can get bold numbers (showing statistically significant better performance) for both M1 vs M2 and M2 vs M1. See, for example, RAPT and MAPT in Table 6. Overall, I think you need to carefully explain what you are doing with the statistical tests.**

Response: In these tables, if the difference is significant, the number will be displayed in bold. For example, the "59" in the top right corner in Table 3 (the original Table 6) indicates that out of 60 scenarios (10 faulty versions × 3 granularities × 2 initial

profiles), RAPT had smaller F-measure than RPT for 59 scenarios. Correspondingly, the "1" in the bottom left corner in Table 3 means that the F-measure of RPT was smaller than that of RAPT for only one scenario. The corresponding null hypothesis was that RAPT and RPT had similar performance in terms of F-measure. Since this hypothesis was rejected, we could say that the fault-detection capabilities of RAPT and RPT were significantly different in terms of F-measure. This statistically significance difference is represented by the bold font of "59" and "1", which further indicates that RAPT was significantly better than RPT. Note that although two numbers ("59" and "1") are given, they are corresponding to one null hypothesis --- each pair of numbers just further tell us the number of scenarios where one technique was better than the other technique.

Action: We have added the relevant content to explain the results of hypothesis testing more clearly in Section 4.1.

R1C8: Did you consider looking at effect size?

Response: Thank you for the recommendation. In the new version, in addition to hypothesis testing, we also calculated the effect size to measure the magnitude of the performance difference between each pair of techniques.

Action: The relevant content and data have been added into the paper (Sections 4.1 and 4.2).

R1C9: As noted, the paper is generally well written. However, I spotted a few minor points:
page 3, add 'of' in 'performance a testing'
page 5, add 'number of' before 'test cases inside c_i' just before S3.4.3
page 6, delete 'for' in 'technique for 20 times'

Response: Thanks for the careful review.

Action: In the revised version, we have made all the revisions as suggested and also checked all the text carefully.

Reviewer 2's comments

***R2C1*: Section 2: A simple explanatory example is missing.**

Response: Thank for the suggestion. An explanatory example greatly improves the readability of the proposed approach.

Action: In the revised version, we have followed the suggestion and added a new section (Section 2.4) to include an explanatory example.

***R2C2*: The biggest part of the paper is the evaluation. Here I have my concerns: Section 3.2: Authors did select faults without disclosing exact selection criteria. The reader does not know which faults they deem "detectable" and which faults were omitted. This unfortunately voids the evaluation.**

Response: Indeed, the original descriptions on the selection of object programs and faults are not clear enough. Actually, we did follow some criteria to select object programs and associated faults for evaluation. As to the selection of object programs, we only included those object programs in the repository that are written in C, whose sizes are larger than 5K LOC in order to make the evaluation as closer as possible to the real software testing, and that are provided with test suites generated using partition testing techniques. Accordingly, we have selected "grep", "make", and "gzip" as object programs. Furthermore, these object programs have different versions, which manifests practical scenarios of software development. As to the selection of faults, we only include those faults that are hard to detect for evaluation; in more detail, we checked all faults in the repository and excluded those faults whose detections require less then 20 randomly selected test cases, except that the detection of the fault in V2 of "gzip" requires about 15 randomly selected test cases. Note that the fault in V3 of "gzip" cannot be revealed by any test cases in the associated test suite and hence it was not included in our experiment. In summary, the empirical studies were conducted on three object programs with ten faulty versions and 19 distinct faults, namely 3 faults for "make", 7 faults for "grep", and 9 faults for "gzip".

Action: In the revised version, we have added a discussion on the selection criteria for both object programs and faults in Section 3.2.

***R2C3*: Table 1: Please explain briefly the meaning of the columns**

Response: Thanks for the suggestion.

Action: In the revised version, we have followed the suggestion and briefly explained all columns in Table 3 (Section 3.2).

***R2C4*: Table 3-5: Give reasons about Number of detected defects and runtime of your evaluation**

Response: Tables 3 to 5 summarize the evaluation results of relevant techniques in terms of F-measure and F2-measure. The results are about the average number of test cases required to detect the first fault or the second fault in different versions of object

programs. They are not about the number of detected defects. In the previous version, we intend to provide the evaluation results with both raw experimental data and boxplot.

Action: In the revised version, we have followed the suggestion of another reviewer and move Tables 3 to 5 and Tables 8-10 to appendix due to the page limit.

R2C5: Tables 8-10 are not really explained, neither are the corresponding box plots.

Response: Thanks for the comment. Indeed, neither Tables 9-10 nor the corresponding boxplots are really explained. This is mainly because they are quite similar to Tables 3-5 and their boxplots.

Action: In the revised version, we have added some explanation based on Tables 8-10 and also the corresponding boxplots in Section 4.2.

R2C6: Table 17-20 could be condensed into one Table to save space.

Response: Thanks for the suggestion. It is really helpful to save space.

Action: In the revised version, we have condensed Tables 17 to 20 into one in Section 4.3.2, and similarly condensed Tables 13 to 16 into one in Section 4.3.1.

R2C7: The evaluated software samples are very similar. For a comprehensive evaluation, different kinds of software need to be evaluated. How would the algorithms perform on larger software components with much more complex interfaces?

Response: We do understand the concern. Although a comprehensive evaluation should include different kinds of software, the development cost of such a comprehensive benchmark prohibits us from doing so. Instead, in our experiment, we have used three object programs from the software artefact infrastructure repository (SIR) and they are popularly used in many studies. This has been clearly claimed as an external threat to validity (in Section 3.6.2), which is also shared by similar studies.

Action: In the revised version, we have added a short discussion about the diversity of object programs in Section 3.6.2.

R2C8: What is the conclusion of the statistical evaluation? Which algorithm is better in which case? What are the conclusions from evaluation? Without a proper evaluation, the use of the algorithm is hard to determine.

Response: In the previous submission, we do answer these questions although the conclusion is not explicitly expressed.

Action: In the revised version, we have added a section (Section 4.4) to conclude the observations from our experiments.

Reviewer 3's comments

***R3C1*: Positive**
- Good introduction and interesting concepts
- Clear information on the research approach
- Many tests done on real programs

Response: Thank you for the endorsement.

Action: None.

***R3C2*: To be improved**
- explain in 3.2 how the faults are located in the various test programs, now only gzip and make are mentioned but the comments are not concise

Response: Indeed, the comments on object programs and faults for evaluation are not concise. Actually, we did follow some criteria to select object programs and associated faults for evaluation. As to the selection of object programs, we only included those object programs in the repository that are written in C, whose sizes are larger than 5K LOC in order to make the evaluation as closer as possible to the real software testing, and that are provided with test suites generated using partition testing techniques. Accordingly, we have selected "grep", "make", and "gzip" as object programs. Furthermore, these object programs have different versions, which manifests practical scenarios of software development. As to the selection of faults, we only include those faults that are hard to detect for evaluation; in more detail, we checked all faults in the repository and excluded those faults whose detections require less then 20 randomly selected test cases, except that the detection of the fault in V2 of "gzip" requires about 15 randomly selected test cases. Note that the fault in V3 of "gzip" cannot be revealed by any test cases in the associated test suite and hence it was not included in our experiment. In summary, the empirical studies were conducted on three object programs with ten faulty versions and 19 distinct faults, namely 3 faults for "make", 7 faults for "grep", and 9 faults for "gzip". ~~Please also refer to our response to R2C2.~~

Action: In the revised version, we have clarified the object program and fault selection criteria in Section 3.2.

***R3C3*: 3.4: why the PT category technique - would this affect the results?**

Response: Partition testing is a mainstream family of software testing technique, which can be realized in different ways, such as *Intuitive Similarity*, *Equivalent Paths*, *Risk-Based*, and *Specified As Equivalent (two test values are equivalent if the specification says that the program handles them in the same way)*. Based on *Specified As Equivalent*, Category Partition Method (CPM) decomposes a functional requirement into a set of categories, which are further divided into disjoint partitions (choices). In this study, we select CPM as candidate partition testing technique because CPM is the most popular one. Such a selection would not affect the results in

general, although it is possible to deliver somehow derivation when different partition techniques are used.

Action: In the revised version, we have added a discussion about this selection in Section 3.4.1.

R3C4: 3.4.3: add the formula from your reference [15] and explain why 0.1 is a fair setting

Response: Thanks for the suggestion.

Action: We have added the formula and given the explanation with 0.1 was the fair setting for γ and τ in Section 3.4.3.

R3C5: 3.6: this section needs an introduction, otherwise a bit "loose" in the paper

Response: Thanks for the suggestion.

Action: In the revised version, we have added a short introduction in Section 3.6.

R3C6: add tables 3,4,5 and 8,9,10 as digital download, you have the boxplots in the paper (+ mention in the headers that a lower score is better)

Response: Thanks for the suggestion.

Action: In the revised version, we have followed the suggestion. We have treated Tables 3-5 and 8-10 as digital download, and also added a short sentence in the header to help the reader understand the meaning of the table.

R3C7: 4.3: explain the holm-bonferroni method briefly

Response: We actually conducted hypothesis testing based on the Holm-Bonferroni method to determine which pair of testing techniques has significant difference in terms of each metric. Across the whole study, for each pair of testing techniques, denoted by technique *a* and technique *b*, the null hypothesis (H_0) was that *a* and *b* had the similar performance in terms of one metric. All the null hypotheses were ordered by their corresponding p-values, from lowest to largest; in other words, for null hypotheses H_0^1, H_0^2, …, we had $p_1 \leq p_2 \leq \ldots$. For the given confidence level $\alpha = 0.05$, we found the minimal index h such that $p_h > \alpha/(N+1-h)$ (where N is the total number of null hypotheses). Then, we rejected H_0^1, H_0^2, …,, H_0^{h-1} that is, we regarded the pair of techniques involved in each of these hypotheses to have statistically significant difference in terms of a certain metric. On the other hand, we considered the pair of techniques involves in each of H_0^h, H_0^{h+1}, .. not to have significant difference with respect to one metric, as these hypotheses were accepted.

Action: We have added the relevant content to explain the details of hypothesis testing more clearly in Sections 4.1 and 4.3.1.

R3C8: 5.0: some overlap with other sections e.g. the constants in DRT; please make it a bit more brief

Response: Thanks for the comment.

Action: In the revised version, we have carefully checked the text and cut redundancy.

R3C9: 6.0 Conclusion is a bit weak, add the good results from 4.1 and 4.2 in here! + add comment on the initial probability profile - this was an interesting result; which could be further investigated.

Response: Thanks for the suggestion.

Action: In the revised version, we have enhanced the conclusion as suggested.