IET Software

推测的和主动的

A SPECULATIVE AND PROACTIVE APPROACH FOR TEST CASE SELECTION: A COST EFFECTIVE ADAPTIVE RANDOM TESTING

SEN-2018-5385 | Research Article

Submitted by: Jinfu Chen, MICHAEL OMARI, Hilary Ackah-Arthur, Patrick Kudjo, Rubing Huang, Jiaxiang Xi

Keywords: SOFTWARE TESTING, SOFTWARE QUALITY, SOFTWARE ENGINEERING

A SPECULATIVE AND PROACTIVE APPROACH FOR TEST CASE SELECTION: A COST EFFECTIVE ADAPTIVE RANDOM TESTING

Michael Omari, *Jinfu Chen, Hilary Ackah-Arthur, Rubing Huang, Patrick Kweku Kudjo, Jiaxiang Xi
(School of Computer Science and Communication Engineering, Jiangsu University, Zhenjiang, 212013, China)
Corresponding author's e-mail:jinfuchen@ujs.edu.cn

Abstract: Fixed Sized Candidate Set (FSCS) is the first of a series of methods proposed to enhance the effectiveness of random testing (RT); collectively referred to as Adaptive Random Testing (ART's) methods. One of the key setbacks of ARTs is their overhead cost in test case generation. Various strategies have been proposed to reduce the overheads of ARTs but regrettably, it has been extremely difficult to balance effectiveness and efficiency. To select a test case, FSCS is caught in a logical trap of evaluating every element of a set of randomly generated candidate test cases by computing their distances to already executed test cases. To reduce the computations, we proposed a new strategy for selecting candidates based on intrinsic property of test cases generated by FSCS. Through data mining, we discovered that distances validated for selecting candidate test cases in FSCS undulates (overlaps) about 50% downward and 50% upward but with a downward net effect. This vital information is used as predictors for discarding further computations once an overlapping condition occurs without vetting every candidate. Experiment and simulation performed indicates that our approach can reduce the overhead cost of FSCS by approximately 25% with negligible adverse effects on effectiveness.

Keywords: Adaptive Random Testing, ART overhead challenge, test case generation, software testing, test strategies.

I. INTRODUCTION

Due to the increasing importance of software in the daily activity of the ordinary person as well as the business community as a whole, the quality of software product has become a major concern to many stakeholders. Software testing has been accepted as one of the most vital component of software engineering and remains one of the most widely practiced and studied approaches for assessing and improving software quality[1]. A software tester is someone who undertakes the testing process with the aim to uncover as many faults as possible given available testing resources. The software is tested by generating inputs (otherwise known as test cases) for execution by the software. There are two common approaches to generating test cases for testing software; black-box and white box. In white box testing, the structure of the software is used as basis for selecting test cases. Black-box on the other hand utilizes no information concerning the structure of the software under test (SUT).[2] Random testing is a very popular black box testing technique which is preferred for its simplicity and intuitive appeal. Random testing selects test cases based on a uniform distribution or according to the operational profile[3]. A test case that reveals a failure is referred to as a failure-causing input [4]. In general, failure-causing inputs determine two basic features of a faulty program. One feature is the failure rate (denoted by θ), which refers to the ratio between the number of failure-causing inputs, and the number of all possible inputs. The second feature is the failure pattern which describes the failure region(s) together with their geometric shape called failure pattern[5]. In random testing test case selected can be replaced or removed from input domain after their execution. The act of selecting and replacing test cases has been criticized because it leads to duplication of test case selection; a waste of resources, but the assumption of replacement is for the purpose of developing simpler mathematical models to aid analysis[3] . It has been reported in [6, 7] that inputs that causes programs to fail turns to form a cluster within the input domain. The distribution and shape of the cluster has been classified as block, strip and point pattern. Figure 1(a)-(c) gives the geometric shapes of the patterns of failure causing inputs. With random testing, the probability of selecting a failure causing input as test a case solely depends on the magnitude of failure rates [8]. This is true for all program patterns, but for block and strip patterns, [8] proposed a slightly modified version of RT called Adaptive Random Testing (ART) which improves the chance of selecting a test case from the failure causing input region. The underlying strategy (recently viewed as a form of diversification[9] of test inputs), works by selecting inputs that are far apart from already executed test cases but which have failed to reveal any program fault. The Fixed-Size Candidate Set ART technique (FSCS-ART), was the first ART technique, and is also the most widely studied[10] was the proposed method based on this intuition.

In the past, failure rate has been the only parameter used for accessing the effectiveness of RT. [8] introduced a new metric called F-measure as a more suitable measure of for evaluating the effectiveness of a testing strategy. It measures the number of test cases which has been executed before the first programs failure is encountered. A very close terminology to F-measure called F-ratio is used to determine the relative effectiveness of ART over RT calculated as the ratio of F-measure (RT) to F-measure (ART). It has been reported in many ARTs [3, 9] [11] [12] that with the exception of point failure pattern which RT has almost similar performance to ARTs in terms of effectiveness, ART outperforms RT, sometimes by very wide margins as 80% [11]when failure pattern is block or strip. Notwithstanding, RT still remains the preferred option for

practical application in software testing tools. This has been largely attributed to the simplicity of the technique and ease of automation[13].

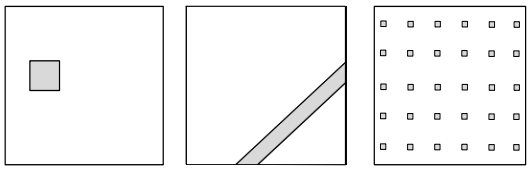

Figure 1 Classification of failure causing inputs
(a) Block Pattern (b) Strip Pattern (c) Point Pattern

II. BACKGROUND

A. FSCS-ART algorithm Description

ART combines randomness of test case generation with even distribution within input domain. Consequently, almost all ART algorithms consist of two independent processes. The first process referred to as candidate generation process involves random generation of program inputs as test case candidates, or briefly candidates. The second process (known as test case identification process) applies some criteria to identify test cases among these candidates to ensure an even spread of test cases across the input domain[5]. The intuition behind this idea is based on the distribution of failure causing inputs within a programs' input domain[8] . In [8] it was reported that, when failure pattern is block or strip, spreading test cases evenly increases the probability of detecting failure with fewer numbers of test cases than ordinary random testing. Therefore improvement in ARTs in terms of overhead cost is very importance since the spreading test cases constitute the most costly facet of the cost of test case generation process.

To ensure optimal spacing among test cases, FSCS maintains two set of test cases throughout testing process; a set of already executed test cases which have not found program fault (referred to as Executed set) and a set (fixed sized) of randomly generated(based on uniform sampling) test cases (referred to as Candidates set). The size of the candidates set is held constant in the entire testing process but the content changes (since they are randomly selected from the input domain) any time a new test case is generated conversely, that of the executed set increases with the size of test suit but content is the same and only changes with an incremental element (the selected candidate) at every round of test case generation. One of the candidates set is to be selected as the next test case based on a criterion. The "best" candidate among the set of randomly generated test cases, is the candidate which is farthest from all the elements within the Executed set.
The "far apart" notion have been variously implemented as maxmin, maxmax and maxsum each requiring that distances between elements of the two sets of test cases be computed in order to select the most suitable candidate.

B. *The overhead problem of FSCS-ART*

FSCS algorithm incurs progressively larger overheads proportional to |E| with a time complexity of $O(n^2)$. Meaning that, to generate the nth test case k $\sum_{i=1}^{n} i = \frac{kn(n+1)}{2}$ number of distances must be computed between k number of candidate test cases and n-1 executed test cases. For two points P and Q; $P = \{p_1, p_2, p_3, \dots, p_d\}$ and $Q = \{q_1, q_2, q_3, \dots, q_d\}$ representing the position of a candidate and an executed test case respectively, in a d dimensional Euclidean space, the distance between them is given by the

The algorithm of FSCS is as follows;

Step 1: Generate the first test case randomly from the input domain using a uniform distribution and goto step 5
Step 2: Randomly generate using a uniform distribution k (a constant) number of test cases and assign as candidate test cases
Step 3: Compute distance (Euclidean) between already executed test cases (E) and the candidate set (C)
Step 4: Select the candidate $c \in C$ which is farthest from all $e \in E$ as the next test case
Step 5: Execute test case
Step 6: If Failure is found goto step 8
Step 7: Add test case to E and goto step 2
Step 8: Exit

Formula:

Distance (P, Q) = $\sqrt{\sum_{i=1}^{d}(P_i - Q_i)^2}$

For a given number of candidate test cases k and executed test cases E, to determine the suitability of a candidate as the test case, the eligibility criterion "farthest candidate" is applied. Consider a candidate set $C = \{c_1, c_2, c_3, \dots, c_k\}$ and already executed test cases $E = \{e_1, e_2, e_3, \dots, e_n\}$ where k is number of candidates, and n represent the already executed test cases. To determine which candidate to be selected for execution, an eligibility criterion given by;

$$min_{i=1}^{|E|} dist(c, e_i) \geq min_{i=1}^{|E|} dist(c_j, e_i) \; \forall c_j \in c \quad \dots \quad (1)$$

This computation must be effected anytime a new test case is to be generated. FSCS approach has been criticized for the expensive nature of its cost overhead. Various attempts have been made in the past to reduce the overhead challenge associated with FSCS. [14]proposed a strategy that seeks to minimize the computational overheads of FSCS by keeping a fixed sized of already executed test cases (referred to as memory) to be used in the distance computation. The implication of this strategy is that part of the executed test cases is ignored (forgotten) in the determination of the "farthest candidate" criterion. Two of this form of forgetting was introduced; Consecutive forgetting and Random forgetting. In consecutive forgetting, the most recent k (memory) executed test cases is used in the determination of the farthest candidate. In contrast, Random forgetting uses k randomly selected executed test cases to search for the farthest candidate. Previous experiment [14] shows that, consecutive forgetting performs better than random

forgetting. Since, the algorithm remembers only a portion of the executed test cases; "half-blind" test cases are generated which adversely affects even distribution. This violation culminates in a poor performance in terms of overall effectiveness. Mao C. et.al [15] proposed a method for dealing with the blind selection problem by introducing a distance-aware forgetting strategy which ensures that a constant number of executed test cases are used in the distance computation process. It applies distance based with grid partitioning to identify neighbouring cells and filter those executed test cases which are within neighbouring cells ("in sight") from those which are beyond neighbouring cells("out of sight"). This leaves a constant number of executed test cases for every candidate distance computation. These strategies are implemented independently of each other. Based on the intrinsic properties of the distribution of test cases generated by FSCS, we extract a predictive variable and use this variable as a basis for a more proactive candidate selection which reduces candidate-executed computations. This approach is easily integrated into any cost reduction approach to FSCS. We explain the details in the preceding section.

III. DISCARD AND CONTINUE (DAC)-RULE

A. *FSCS test case distribution*

As indicated earlier the selection of a test case by FSCS involves random selection (candidates) as well as even distribution (by selecting farthest candidates from already executed test cases). Following the original proposal of FSCS we adopt the maxmin criteria in test case selection. The random nature of selecting candidates means that, the maxmin criterion does not always generate test cases with consistently declining distances (maxmins) between executed test cases. It is however predictable that in the long run maxmin distances reduces with increasing number of executed test cases.

为什么

In order to fully understand the behaviour of the maxmin for selected test cases we mined the simulated data generated by FSCS. Figure 2 is a plot for the simulated data. Intuitively, the distance drops at a very fast rate then later reducing at a gradual rate and eventually becoming asymptotical with maxmin=0.0. The overall trend shows a continual decline in the distances among test cases as is expected as the input space gets crowded. Figure 3 provides a snapshot of the first 200 test cases and the distribution more closely showing differences in the actual distances which were validated for selecting test cases in FSCS.

为什么不找一个真实的程序来观察maxmin的值？

The points on the graph represent differences between consecutive selected candidates' maximum-minimum distances for the nth test case generation. It can be observed that there are almost equal instances where $MaxMin_n > MaxMin_{n-1}$ (n=the nth test case generation) depicted by the points lying above the 0.0 margin whiles those below the 0.0 margin represents the opposite case. The differences between consecutive maxmin therefore do not show a continual declining trend within points around the nth test case generated but undulates over a number of consecutive

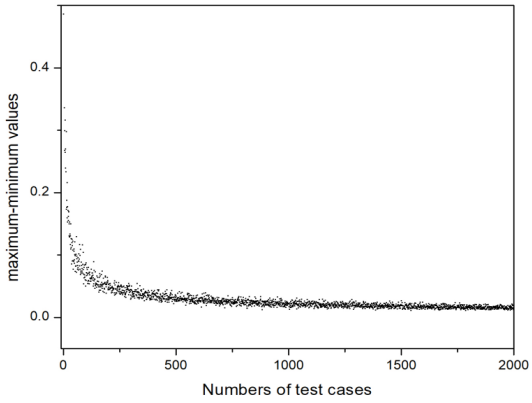

Figure 2. Test cases generated and their corresponding maxmins which were validated for their selection

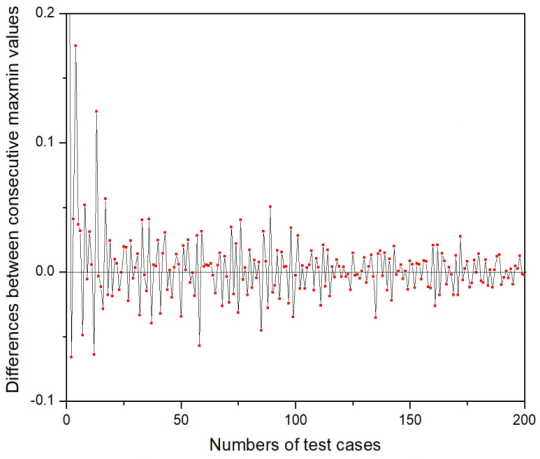

Figure 3. Test cases generated and the differences between their maxmin

做一个对比图啊

test cases. Having analysed the magnitudes of the overlapping distances, we observed that the long term trend (the net effect) shows that the negative (downward) magnitude outweighs the upward magnitudes (i.e. net effect is negative) as can be seen in the declining of deviations around 0.0 in figure.3 and the asymptotic behaviour in figure 2). However, around a specific test case the maxmins' overlaps.

B. *Predicting the maxmin values*

We introduce a new form of forgetting strategy referred to as discard and continue (DAC)-rule to reduce the number of distance computations of FSCS that is based on the behaviour of the maxmin distances. In FSCS distance computation must be effected between each candidate in C and all executed test cases in E irrespective of whether that distance was determined 'earlier' or not. Maxmin criteria requires that all candidate must be evaluated to arrive at the best (valid) candidate.

In DAC-rule, we ignore (forget) further distance computation between candidates set and executed set if minimum distance of current candidate is greater than previously computed maxmin. It is logical, that as the input space gets crowded with increasing number of executed test cases, distance between already executed test cases will decline in proportion to |E|. Hence, a candidate whose minimum distance is greater or equal to that of previous maxmin can be considered good predictors of the actual maxmin distances, far enough for execution. Previously, this piece of information has not been utilized.

Let's say in a one dimension input domain, the maxmin for the 6th test case was 8 , and the already executed test case set is E={11,81,55,29,70,85} then to generate the 7th test case, 10 candidates C={16,4,20,7,50,23,6,18,12,33) were randomly selected, the maxmin for the currently selected candidates is 9 given by candidate 3 (i.e. c3= 20) FSCS must go through computing the minimum distance between each of the ten candidates before finally settling on candidate 3. We propose that, after the 3rd candidate was evaluated giving a value of $9 > 8$ (the maxmin for the 6th test case), the rest of the computations for the 4th to the 10th candidate should be discarded. Suppose for the purpose of argument, we assume the candidates selected was {16,4,20,7,50,23,6,18,45,33}, it can be observed that candidate 9 (value of 45) has the highest minimum distance of 10, however we argue that since 9 is greater than the previous maxmin value (7), it suffice to settle on the 3rd candidate because it is "far enough".

As observed in Figure 3, anytime we skip the margin before reaching the next point, an overlapping condition has occurred. The distances computed for the points below and points above represent the overlapping distance (used as predictors). The previous maxmin distance becomes the predictor for the current maxmin distance. This indicates that a substantial portion of distance computations of FSCS can be avoided without compromising much on the effectiveness of the method. It can easily be inferred that, the absolute savings in time cost given the same failure rate, will increase proportionately with the dimension of input domain since number of computations increases proportionately with dimension.

The best case scenario is, if the first candidate selected meets this criteria whiles the worst case scenario is, when the last candidate/none of the candidates meets this criteria. However, the trade-off is favourable since the only effort in the worst case is k-σ number of comparisons where σ is the number of instances where minimum distance of current candidate is less than minimum distance of previous candidate(s). In a single case of a particular candidate meeting this criteria say, on the τth candidate in the nth test case generation, potentially $n \times (k - \tau)$distance computation is avoided.

If we encounter a candidate whose minimum distance is greater than that of the previous maxmin, we settle on the candidate 'in hand'. Otherwise, we assign the current maxmin as the predictive value for the next maxmin distance. Given two consecutive test cases t_{n-1} and t_n, if maxmin t_n > maxmin t_{n-1}, we say that, the maxmin distances overlaps.

The results from the mined data placed the number of overlapping instances at 50.02% ≈50% of total computed maxmin. By probability, the average (k- τ)th candidate is approximately in the middle, halve of the candidates-executed distance is wasted on searching for the best candidate which is already "in hand", and about halve of the time the maxmin distance of previously executed test case is greater than the maxmin of the current best candidate, theoretically, 50% of these 50% (25%) distances computations can be discarded.

We present a modified FSCS algorithm as follows

ALGORITHM FOR DAC-FSCS

```
1: Use FSCS to generate the first two test cases t_1 and t_2
2: Set E= {t_1, t_2} and Set Previous_Farthest =dist (t_1,
   t_2) // maxmin distance computed in step 1
3: Set maxmin =0, min=0;
4: for i=1 to k //k=number of candidates set
5:      Randomly generate test case from input domain
      (according to uniform distribution)
6.      Add test case to C;
7: end for
8: for each Candidate c in C
9:         Compute Euclidean distance between C and its
           nearest neighbour in E and assign to min
10:        if min>maxmin
11:             maxmin=min;
12:             Farthest_Candidate=c
13:             if maxmin >Previous_Farthest
14:                  Previous_Farthest=maxmin
15:                  goto step 20
16:             end if
17:        end if
18: end for
19: Assign maxmin to Previous_Farthest
20: execute Farthest_candidate
 21: if failure detected goto step 24
22: Add Farthest_candidate to E
23: goto: step 3
24: return |E| and exit
```

C. *Analysis of candidate test case selection and the impact of DAC-rule*

In the simulation exercise a total of 5000 test cases were generated using FSCS (k=10) without DAC-rule and the same numbers were generated using FSCS with DAC-rule (DAC-FSCS). Figure 4 represent the frequency of candidate selection for FSCS with and without DAC-rule. It can be observed that there is no specific pattern in terms of the selection preferences of candidates as test cases. This is because each candidate has equal probability of being selected and therefore the frequency of each candidate selected falls just around the mean 500 with almost the same numbers of candidates (small variant) selected above and candidates below the mean score in a random fashion. The graph of FSCS with DAC-rule shows a completely different outcome with a clear pattern of declining number of

frequencies for candidates' selection as we move from candidate 2 through to candidate 10.

It shows that preferences were given to earlier candidates in the selection of candidates as test cases. It can be observed that the first five candidates had frequency greater than the mean frequency for the entire candidate set. This is evident that the FSCS with DAC-rule uses fewer numbers of candidates in the distance computation process than FSCS. The closer the selected candidate is to 10, the less efficient DAC-rule becomes vice versa. Fewer frequencies for latter candidates mean better efficiency.

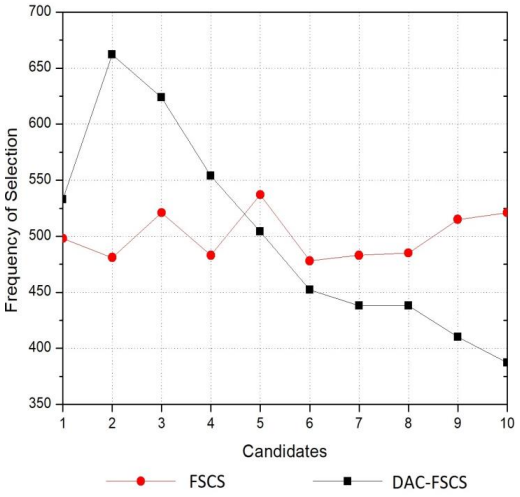

Figure 4. Candidates and their selection frequencies

It also shows that many of the latter candidates were discarded off (forgotten) after the overlapping condition was satisfied before computation could reach those candidates. This outcome is not surprising because as we traverse through the first candidate to the next, the probability of finding a suitable candidate increases which makes sense from statistics point of view.

Supposed in the determination of a suitable candidate for the nth test case, an overlapping condition occurs at candidate C_i, then, the number of discarded computations for the nth test case generated is $n \times (k - C_i)$ where k is the number of candidates. Supposed again that after the executed test cases reaches φ the number of overlapping conditions is α then total number of discarded computations is given by

$$N = \sum_{i=1}^{\alpha} \{(K - C_i).|E|_j\}$$

Where $|E|_j$ represents the size of E when $\alpha = i$.

In comparison with FSCS, the number of computations with DAC-rule when test suit reaches φ is given by:

$$N_{DAC-FSCS} = K \sum_{i=1}^{\varphi} i \Big|_{FSCS} - \sum_{i=1}^{\alpha} \{(K - C_i).|E|_j\} \Big|_{DAC-rule}$$

C_i is the candidates upon which the overlapping condition occurred.

The ratio of the number of discarded computations to the number of test computations by FSCS is

$$\xi = \frac{N}{K \sum_{i=1}^{\varphi} i|_{FSCS}}$$

ξ represents the efficiency ratio. In order to determine the value of the parameter ξ we performed multiple simulations using various dimensions (the number of dimensions influences the values of N) and generating 10,000 test cases in each simulation. By taking the averages across the various dimensions, the value of ξ is estimated to be approximately 0.2464301 which is just close to 25%.

D. Complexity of DAC-FSCS

The percentage of computations discarded ξ is approximately ¼th of the original FSCS and therefore if we consider the complexity of FSCS which is determined by the nth test case generated as k $\sum_{i=1}^{n} i = \frac{kn(n+1)}{2}$ then after the introduction of the DAC-rule, the computations reduces to

$$\approx k \times (1 - \xi) \sum_{i=1}^{n} i = \frac{kn(n+1)}{2} \times (1 - \xi) = O(n^2)$$

which is still quadratic in terms of the number executed test cases since |E| becomes the dominant part of the test case generation.

IV. RESEARCH QUESTIONS

We have proposed a new method for reducing the cost overhead of FSCS. Often times efforts at downsizing cost ends up in compromising effectiveness. We are therefore interested in how our proposed method improves upon efficiency and how it impacts on effectiveness. So we ask "is this also the case with FSCS with DAC-rule (henceforth DAC-FSCS)?' and if so, to what extent?

We examine these questions in the light of FSCS itself and then other forgetting strategies including consecutive forgetting) and random forgetting; (here after C-FSCS and R-FSCS respectively) adopted from the convention used in [15]. Our research questions are therefore divided into two broad perspectives and four key questions;

RQ1: How effective is DAC-FSCS compared to FSCS?
RQ2: How efficient is DAC-FSCS in relation to FSCS?
RQ3: How does it compare to forgetting methods R-F-FSCS in terms of efficiency?
RQ4: How effective is DAC-FSCS compared to other cost reduction strategies of FSCS?

In order to find answers to our research questions, we performed a series of simulations and experimentation exercise.

A. SIMULATIONS

In the simulation exercise, we mimic a real program by generating programs parameters including the input domain, failure rate and failure region. The input domain and failure rate is predefined before the testing process begins, whiles the failure region is randomly placed in the input domain anytime the testing process is reinitiated.

The simulation allows us to look at the performance of our proposed strategy in a much broader perspective. We used the simulations to evaluate our proposed method under different scenarios in a much more comprehensive detail by varying failure rates, failure patterns and locations within the input domain.

To access effectiveness of ARTs, F-measure has been used in the past since it directly expresses the quality of test input generated by the testing strategy [8]. Moreover, it has been used extensively in many ART papers[8, 16] as an effectiveness evaluation metric. Accordingly we make use of same metric to facilitate comparative analysis. Since test cases generated by ART exhibit a semblance of randomness, it is difficult to determine an accurate value for F-measure for a method in a particular program with just a single test run. Moreover no theoretical framework exists for estimating the f-measure of an ART algorithm. To this end, multiple tests run needs to be performed in order to arrive at a statistically reliable mean value for the F-Measure within a certain confidence interval and a defined error margin. In our paper, we used 95% confidence interval with +-5% margin of error to arrive at the mean F-measure values.

1) Answer to RQ1- How effective is DAC-rule compared to FSCS?:

First we were interested in finding out how effective DAC-FSCS is in relation to FSCS across different failure rates under different pattern of failure regions. Table 1 shows the results for DAC-FSCS and FSCS in three (3) failure patterns; block, strip and point. It can be observed that DAC-FSCS has similar behaviour as FSCS in all the failure patterns in terms of performance. It scored around 64% on the block, 85% on the strip and 96% in the block patterns.

In general irrespective of the failure rate and pattern there is consistency in the trend of performance of DAC-FSCS in relation to FSCS. It can be observed that the F-measure of FSCS is a little better than DAC-FSCS, however the difference is mostly around than 1% which is very small.

提的方法没有用啊

2) RQ2: How efficient is DAC-FSCS in relation to FSCS? :

In this question we considered to the relative efficiency of DAC-FSCS to FSCS and whether the improvement rate was consistent across different dimensions of input domain.

This question subsumes two relevant questions; how efficient is the method in itself with no reference to FSCS and how effective is the method in relation to the overall cost reduction in FSCS. As stated earlier, theoretically, DAC-rule can reduce computations of FSCS by as much as 25%, we verify to what extent this can be true in practical terms.

Table 1 Performance comparison of DAC-FSCS and FSCS under different failure rates and failure pattern of simulated programs

	BLOCK PATTERN		**STRIP PATTERN**		**POINT PATTERN**	
Failure rate(θ)	**FSCS**	**DAC-FSCS**	**FSCS**	**DAC-FSCS**	**FSCS**	**DAC-FSCS**
0.01	68.78%	68.89	93.14%	93.21%	100.13%	99.47%
0.005	65.76%	65.97	93.75%	93.77%	99.68%	99.69%
0.002	64.15%	64.23%	96.39%	96.41%	97.46%	97.49%
0.001	63.81%	63.93%	97.23%	97.42%	98.96%	99.21%
0.0005	63.42%	64.44%	97.22%	97.24%	97.89%	98.92%
0.0002	62.55%	63.16%	98.24%	98.25%	97.66%	97.67%
0.0001	61.84%	62.17%	99.32%	99.82%	97.18%	98.22%

Table 2 comparison of DAC-FSCS and FSCS in both F-measure and FM-time for selected dimensions with a block failure pattern

	F-Measure		**%Loss of**	FM-time		**% Gain in**
D	DAC-FSCS	FSCS	**Effectiveness**	DAC-FSCS	FSCS	**Efficiency**
1	563.0	570.2	0.889	558.0	704.37	20.73
2	639.8	637.1	0.430	1404.	1795.4	21.75
3	754.8	753.0	0.243	3597.	4614.3	22.04
4	902.5	906.1	-0.392	6744.	8846.1	23.75
10	2747	2742.	0.135	86212	116873	26.47

Table 3 Computational overhead comparison for DAC-FSCS, FSCS and other forgetting methods (m=50)

	Computational time in Seconds						
Algorithm	N=100	N=250	N=500	N=1000	N=2500	N=5000	N=10000
FSCS	0.00061	0.00313	0.01038	0.6081	5.6305	11.8383	46.4203
DAC-FSCS	0.00032	0.00157	0.00626	0.3486	3.7730	8.1896	36.6064
R-FSCS	0.00248	0.00750	0.03452	0.9374	0.2816	0.39461	0.66218
C-FSCS	0.00193	0.00432	0.09654	0.8984	0.0810	0.23490	0.43281

Table 2 shows the performance of DAC-FSCS as compared to FSCS in dimensions 1, 2, 3, 4, and 10 with a failure rate of 0.001. From the results obtained DAC-FSCS has similar performance in the F-measure values in all the dimensions under study however when it comes to the fm-time, DAC-FSCS much better performance over FSCS. The magnitude of the gap in the fm-time increases with the dimension of input domain as expected. The percentage improvement also moves in almost similar fashion. It can be observed that, the improvement rate increases as well with increase in dimension from 20% in 1dimension to 26% in 10 dimensions. The overall average improvement is about 23% across the five (5) selected dimensions. Figure 4 is a graphic representation of the cost overhead between FSCS and DAC-FSCS. Even though both function is quadratic in complexity, the fm-time of DAC-FSCS lies below that of FSCS for every number of test cases generated.

1) Answer to RQ3: How does it compare to other forgetting methods in terms of efficiency?:

Computational overheads comparison

The main research question is about how efficient DAC-FSCS algorithm is in comparison with FSCS and other forgetting methods so we dedicated an experiment purposely to determine the actual running time of the various methods and compare with that of DAC-FSCS. This is essential because complexity analysis alone may mask certain important constants in practical scrutiny[8] such as ξ discussed in section II.

From table 3 it can be seen that, the computational overheads of DAC-FSCS is initially better than R-FSCS and C-FSCS for number of test cases less than 1000. When N reaches 1000 and beyond, R-FSCS and C-FSCS had a low overhead cost since their complexities are linear compared with the quadratic time of DAC-FSCS. When N reaches 10000 the time for the other forgetting methods is averages about 5 seconds whiles that of FSCS and DAC-FSCS reaches 46 and 36 seconds respectively.

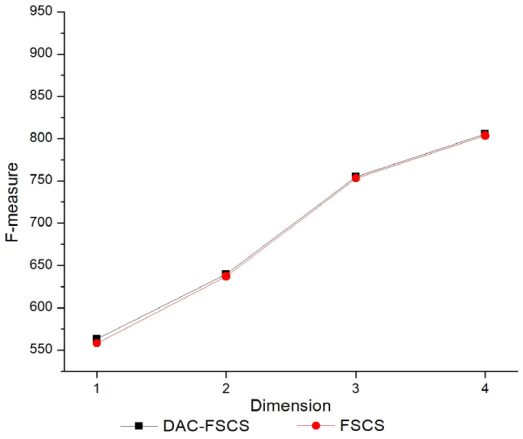

Figure 5. F-measure for FSCS and DAC-FSCS in a simulated programs for some selected dimensions

We performed a simulation to compare the overhead cost of FSCS and DAC-FSCS. The result of our simulation is presented in figure 6. It can be observed that, even though both methods have a quadratic time complexity with respect to size of test suit, DAC-FSCS is a fraction of FSCS in the head to head comparison on each number of test cases generated. R-FSCS and C-FSCS on the other hand are both linear time complexity with C-FSCS a little better than R-FSCS.

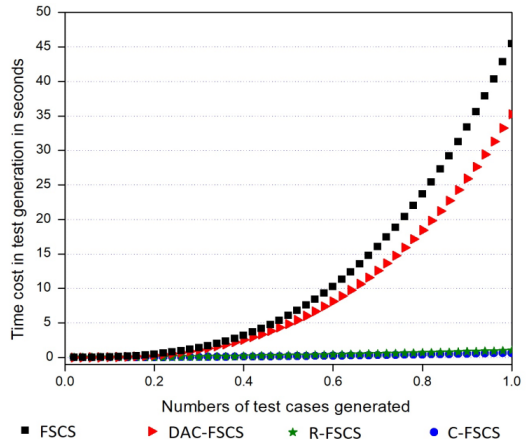

Figure 6. Overhead comparison of four methods under consideration

2) Answer to RQ4: DAC-FSCS compared to R-FSCS and C-FSCS?:

How is DAC-FSCS compared to R-FSCS and C-FSCS methods in terms of effectiveness? In this research question, we were interested in the difference in failure detection effectiveness of DAC-FSCS in relation to other forgetting algorithms.

The table 4 shows the performance of DAC-FSCS against two other forgetting methods (C-FSCS and R-FSCS) in three failure patterns under different failure rate ranging between 0.01 and 0.0001. It can be observed that C-FSCS and R-FSCS have very poor performance in the block failure pattern compared to DAC-FSCS in all the failure rates. The difference in the mean values is significant (P-values<0.005) for all values of θ. In the strip pattern, the performance of C-FSCS and R-FSCS still fell behind DAC-FSCS but with a marginal rate compared to the block pattern. When we carefully analysed the performance against the failure rate, it can be observed that, as the failure rate increases the margin of the difference drops gradually in the case of the strip and point pattern. The difference between the population means of DAC-FSCS and C-R-FSCS is not significant (P-values >0.005) in the strip and point pattern when θ >0.002.

Table 4 Performance comparison under block pattern

BLOCK PATTERN				
Failure rate(θ)	R-FSCS	P-value	C-FSCS	P-value
0.01	-25.3%	0.000	-14.8%	0.000
0.005	-39.4%	0.000	-30.97	0.000
0.002	-41.7%	0.000	-41.2%	0.000
0.001	-43.2%	0.000	-40.13%	0.000
0.0005	-42.4%	0.000	-42.94%	0.000
0.0002	-42.5%	0.000	-40.16%	0.000
0.0001	-42.8%	0.000	-43.17%	0.000
STRIP PATTERN				
0.01	-7.20%	0.0000	-5.21%	0.000
0.005	-3.75%	0.0021	-4.97%	0.000
0.002	-3.39%	0.0038	-5.61%	0.000
0.001	-3.23%	0.0022	-3.42%	0.005
0.0005	-4.22%	0.0025	-2.94%	0.023
0.0002	-2.24%	0.0394	-2.65%	0.143
0.0001	-1.92%	0.0639	-1.02%	0.351
POINT PATTERN				
-5.13%	0.0000	-5.47%	0.000	-5.13%
-2.68%	0.0342	-3.69%	0.0590	-2.68%
-3.46%	0.0071	-2.49%	0.0743	-3.46%
-1.96%	0.2653	-0.21%	0.6328	-1.96%
-0.89%	0.3764	-0.92%	0.4321	-0.89%
-2.66%	0.0628	-1.67%	0.2875	-2.66%
-1.28%	0.7435	-0.72%	0.5823	-1.28%

Table 5 Characteristics of failure pattern of experiment programs

Program	Failure Pattern
Airy	A block in the centre of input domain
Erfcc	A block in the centre of input domain
Probks	A block in the centre of input domain
Tanh	A block in the centre of input domain
Bessj0	A block in the centre of input domain
Bessj	strips
Gammq	A long narrow strips
Sncndn	Points scattered over the entire input domain
Golden	Points scattered over around a very large hyperplanes
Plgndr	Strips near the edge of the input domain
Cel	One failure region on the whole edge side of the input domain
El2	Strips near the edges

Table 6 Experimental programs and their feature

Program	D	**Input Domain**		**Seeded fault type**				**Total**	**Failure rate**
		FROM	TO	AOR	ROR	CR	SVR		
Airy	1	-5000	5000				1	1	0.000716
Bessj	2	(-2.0,-1000)	(300.0,15000.0)	2	1		1	4	0.001298
Bessj0	1	-300000.0	300000.0	2	1	1	1	5	0.001373
cel	4	(0.001,0.001,0.001,0.001)	(1.0,300,10000,1000)	1	1		1	3	0.000332
el2	4	(0.0,0.0,0.0,0.0)	(250,250,250,250)	1	3	2	3	9	0.000332
erfcc	1	(-30000.0)	(30000.0)	1	1	1	1	4	0.000690
gammq	2	(0.0, 0.0)	(1700.0, 40.0)		3		1	4	0.000574
golden	3	(−100.0,-100.0,100.0)	(60,60,60)		3	1	1	5	0.000830
plgndr	3	(10.0,0.0, 0.0)	(500.0, 11.0, 1.0)	1	2		2	5	0.000368
probks	1	(-50000.0)	(50000.0)	1	1	1	1	4	0.000387
sncndn	2	(-5000.0,-5000.0)	(5000.0,5000.0)			4	1	5	0.001623
tanh	1	(-500)	(500)	1	1	1	1	4	0.001817

B. EMPERICAL STUDIES

Having accessed the performance of DAC-FSCS in simulations, we intended at this point to see if empirically, the results obtained can be verified using real world programs. Real world programs which were used in our experiments were 12 in number of different dimensions ranging from one (1) to four (4) and different failure rates and types of faults inserted. These programs have been used extensively in the past [3, 8, 11, 15, 17] to assess the effectiveness/efficiency performance of most ART algorithms in order to put the performance into proper perspective. These programs were originally written in Fortran Pascal or C containing 30 to 200 statements[11]; designed for numerical computations and have been converted to C++ programs. Table 5 provides the characteristics of the failure pattern of the programs. Table 6 provides some essential features about the programs including their names, failure rates, number of seeded faults, and the type of fault seeded. The seeded error types includes; Arithmetic Operator Replacement (AOR), Relational Operator Replacement (ROR), Constant Replacement (CR) and Scalar variable Replacement (SVR).

1) Answer to RQ1: How effective is DAC-FSCS compared to FSCS?:

The table 7 shows the ANOVA test results of our study on the twelve experiment programs for FSCS and DAC-FSCS. It can be observed that, there is a very marginal difference in the F-measure values of both DAC-FSCS compared to FSCS. In most cases, the p value is >0.005 indicating that there is no significant difference in the means score of the two methods in most of the programs. Specifically, 8 out of 12 programs showed no significant difference whiles 4 out of 12 showed the opposite case but the absolute values in their overall mean showed a very marginal difference. Table 8 shows the actual recorded values of their performance in all 12 experiment programs. As presented in the table, the programs cel recorded the worst performance with a drop in effectiveness by 65 representing 4% of FSCS results. This is followed by El2 with a similar percentage drop. For the programs golden and sncndn DAC-FSCS performed better than FSCS but the difference is not very significant.

2) Answer to RQ2: How efficient is DAC-FSCS in relation to FSCS?:

Table 8 shows the performance of both FSCS and DAC-FSCS in terms of the number of test cases used to find the first fault and the time interval within which it occurred. It can be seen that while the f-measure values recorded for DAC-FSCS shows a very marginal decline in the values recorded for that of FSCS, there was some significant gains in the time taken to discover the first fault (fm-time). Most of the programs recorded fm-time values ranging from 19% (in bessj0) to 26% (in Plgndr) less than that of FSCS. On the average 22.2 % of time has been saved by DAC-FSCS when compared to FSCS.

3) Answer to RQ4: DAC-FSCS compared to R-FSCS and C-FSCS

Table 9 is a description of the statistical data of ANOVA detailing the performance of DAC-FSCS, R-FSCS and C-FSCS. The result shows that DAC-FSCS have a far superior performance in terms of F-measure. In all the programs under consideration, with the exception of golden and sncndn, C-FSCS fell behind DAC-FSCS in the rest of the programs. The ANOVA test shows that there is a significant ($p > 0.005$) difference in the means values for ten (10) out of twelve (12) of the experiment program. Specifically with the one dimensional programs, the performance of C-FSCS is very poor. In the case of two programs sncndn and golden C-FSCS performed better than DAC-FSCS but there is very little difference in the F-measure score (mean difference recorded is just 31.76 and that of sncndn 2.96845). This may be attributed to the facts that, the failure pattern in these programs are of point type. An experiment conducted by [8] has showed that, even distribution of test cases does not have any significant impact on failure detection abilities of ARTs when failure pattern is of a point type.

Table 7 ANOVA test on F-measure between FSCS and DAC-FSCS.

Prog.	FSCS	DAC-FSCS	Mean Diff	SIG.
bessj0	401	406	2.4	0.61859
probks	1535	1535	-0.30723	0.96126
airy	798	795	-3.56475	0.28902
plgndr	1613	1616	2.9635	0.80541
golden	1876	1878	2.38308	0.91066
erfcc	977	965	-12.05945	0.09683
sncndn	627	629	2.60695	0.71457
tanh	315	315	-4.73827	0.00187
cel	1605	1537	-65.38119	9.24083E-8
el2	736	714	-31.7974	4.39617E-6
gammq	1075	1080	-4.11434	0.34774
bessj	451	451	-7.56075	0.01049

Table 8 Effectiveness and Efficiency trade-off between FSCS and DAC-FSCS in experimental program.

Program	**F-measure**		%Loss of Effectiveness	**Fm-time** % Gain in Efficiency
	FSCS	**DAC-FSCS**		
bessj0	401	406	0.012125	0.197634
probks	1535	1535	0.000262	0.246789
airy	798	795	0.004464	0.204436
plgndr	1613	1616	0.042605	0.266513
golden	1876	1878	-0.00127	0.222034
erfcc	977	965	0.012334	0.244907
sncndn	627	629	-0.00414	0.245022
tanh	315	315	0.014793	0.199112
cel	1605	1537	0.040787	0.206817
el2	736	714	0.042605	0.216513
gammq	1075	1080	0.003782	0.224547
bessj	451	451	0.016488	0.190831

Table 9 ANOVA test on F-measure between DAC-FSCS and two other forgetting methods.

	C-FSCS		R-FSCS	
Program Name	Mean Difference	SIG	Mean Difference	SIG.
bessj0	-274.495	2.02113E-8	-269.06255	2.02113E-8
probks	-1216.79	2.02113E-8	-1226.171	2.02113E-8
airy	-609.946	2.54629E-8	-645.47345	1.0461E-7
plgndr	-998.599	2.02113E-8	355.059	2.02113E-8
golden	31.76	0.17552	-22.8205	0.32862
erfcc	-723.268	2.02113E-8	-773.9874	2.02113E-8
sncndn	2.96845	0.66191	4.01755	0.55668
tanh	-246.822	2.02113E-8	-235.71555	2.02113E-8
cel	-1380.69	2.02113E-8	-901.64143	4.53035E-8
el2	-645.260	1.86931E-8	8.24048	0.37726
gammq	-118.04	2.03771E-8	124.92456	1.90809E-8
BESSJ	-292.023	2.02113E-8	-200.15815	2.02113E-8

Similar to C-FSCS, R-FSCS's performance is poor in most of the 12 experiment program with a very high margin in the mean score of F-measure but the R-FSCS performance remains stable compared to DAC-FSCS in the programs sncndn, plgndr, gammq and golden. It performed better than DAC-FSCS in programs plgndr and gammq with a significant ($p>0.005$) difference in the mean scores values.
On the other hand R-FSCS performance is worse in the one dimensional programs erfcc, probks, and tanh. At a 95% confidence interval, there is a significant difference ($p> 0.005$) in the mean values for eight (7) out of the twelve (12) real-life programs of which DAC-FSCS performed better than R-FSCS.

V. RELATED WORKS

The overhead cost of ARTs has been a major challenge since its early inception. Some efforts have been made to counter this challenge in the past. A method like MART was proposed in the earlier in the introduction of ART to reduce the overhead cost of ART. Earlier studies on MART[16, 18] have showed that MART can achieve similar performance as FSCS with lower cost. Efficiency of MART is dependent on the number of domains within the input space assigned as mirrors i.e. $O(n^2/m^2)$ where m is the number of mirror domains in a selected MART scheme. Studies[17] showed that MART performs abysmal when failure causing inputs are not related to some dimensions of the input domain. In [17], Dynamic Mirror Adaptive Random Testing (DMART) has been proposed as a solution to this challenge with better performance over MART especially in high dimensions programs simulated. The complexity of DMART is linear in terms of the number of executed test cases within partitions after it has reached a certain threshold (determined by the probability that there exist at least one test case in each partition) which triggers a Dynamic Mirror Partitioning (DMP) function.

Besides the mirroring strategies, forgetting strategies have also been proposed to minimize the distance computations of D-ARTs. The earlier methods of forgetting such as Consecutive forgetting and Random forgetting have been effective at reducing computational requirement for ARTs from quadratic to linear complexity since the cost of generating a new test case no longer depends on $|E|$ but is based on a predefined constant (memory parameter). However, as pointed out in[14] simulations results shows that the F-measure of the ART algorithm with forgetting depends on the size of the memory used as representative of the executed set: The fewer executed test cases were used, the larger the F-measure was.

In order to alleviate the blind selection challenge of the methods, Mao C. et.al [15]proposed a forgetting approach using distance-based and partition base ART approach. By using grid partitions, neighbouring cells are computed and the number of executed test cases that lies in the neighbouring cells determined. The executed test cases which are "out of sight" of candidates test cases are those test cases which are outside the neighbouring cells. The unforgotten test cases are those that remain within candidate and its neighbouring cells leaving a constant number of executed cases that are used in distance computations. An upper bound is placed on the number of executed test cases in sight in order to control the number of computations. The complexity of test case generation in [12] is of linear order while providing comparable effectiveness to FSCS.

In addition to the distance based approaches for enhancing efficiency of ART; other approaches are based on partitioning of input domain. Partition ART approaches [12, 18, 19] have been introduced for generating test cases with low overheads compared to distance based approaches. A hybrid of partition and distance methods such as divide and conquer[20] have been proposed to reduce time complexity of ART by repeatedly partitioning the input domain once a threshold of executed test cases in each partition is reached. The cost of test case generation is the number of distance computation between candidates set and the threshold parameter of executed test cases in each partition of input domain.

In[21] Shahbazi et al. proposed a new algorithm that seeks to spread random test cases evenly in input domain by using Centroidal Voronoi Tessellations (CVT). Random Border Centroidal Voronoi Tessellations (RBCVT) is proposed (with quadratic complexity in order of $O|E|^2$) and its linear runtime (in order of $O|E|$) version RBCVT-Fast based on a novel search algorithm. It is not an independent approach for test case generation hence it requires the use of other ART methods to generate initial points.

Different from the approaches presented above, we have proposed a new method for reducing the overhead cost of FSCS which is based on the intrinsic properties of the distribution of test cases generated by FSCS. Our method addresses the underlying weakness of a distance-based ART based on data collected from already generated test cases. It is therefore applicable to all proposed ARTs which are based on FSCS and all methods proposed for reducing complexities of distance-based ARTs including those mentioned in the literature.

VI. CONCLUSIONS

ART is an effective approach for finding program fault as compared to RT. However, the underlying cost for implementing ARTs often degenerate into inefficiency in performance due to the extra computations used to guide the test case selection process. For instance, the algorithm for FSCS has a quadratic complexity; i.e. $O(n^2)$ hence test cases generation time increases significantly with increasing number of unsuccessful failure detection. Many efforts have been geared towards reducing the efficiency deficit of ART but have been unsuccessful at maintaining the performance

level of the methods in terms of effectiveness. We have proposed a light weight method for improving the efficiency of FSCS with minimal "side effects" on effectiveness which at the same time easy to incorporate in other overhead reduction methods of FSCS. The approach works by curtailing distance computations based on the values of a predictive variable, maxmin distance. Anytime we encounter an instance where the minimum distance of the current computation (candidate) is greater than previously computed maxmin, we discard the rest of the computation process to settle on that candidate. Theoretically our approach is able to reduce the overhead cost of FSCS by about 25%. Simulations and experimental studies were curried out to check the veracity of the theory empirically. The results were largely consistent with the theoretical analysis. The statistical analysis of the results showed insignificant difference in the means of the F-measure values recorded in most of the experiment programs. To examine the strength and weaknesses of our proposed method, we have compared it with two forgetting methods. In the experimentation and simulations aspect of our study, our method's performance in terms of effectiveness is comparable to that of FSCS. The run-time complexity of the newly proposed method DAC-rule is still quadratic just like FSCS. Despite the quadratic complexity of the proposed method, the computational cost of about 25% less when compared to the original FSCS approach which is very significant. It is also a very flexible but effective way of reducing the computational cost of FSCS. In the light of this, we recommend it as the base substitute method in place of FSCS since its comparative effectiveness is not significantly different.

REFERENCES

1. Orso, A. and G. Rothermel. *Software testing: a research travelogue (2000–2014)*. in *Proceedings of the on Future of Software Engineering*. 2014. ACM.
2. White, L.J., *Software testing and verification*, in *Advances in computers*. 1987, Elsevier. p. 335-391.
3. Chan, K.P., T.Y. Chen, and D. Towey, *Restricted random testing*, in *Software Quality—ECSQ 2002*. 2002, Springer. p. 321-330.
4. Budd, T.A., *Mutation analysis: Ideas, examples, problems and prospects.* Computer Program Testing, 1981. **8**: p. i29-l48.
5. Chen, T.Y., F.-C. Kuo, and H. Liu, *Application of a failure driven test profile in random testing.* IEEE Transactions on Reliability, 2009. **58**(1): p. 179-192.
6. Chen, T.Y., T. Tse, and Y.-T. Yu, *Proportional sampling strategy: a compendium and some insights.* Journal of Systems and Software, 2001. **58**(1): p. 65-81.
7. Ammann, P.E. and J.C. Knight, *Data diversity: An approach to software fault tolerance.* Ieee transactions on computers, 1988(4): p. 418-425.
8. Chen, T.Y., H. Leung, and I. Mak. *Adaptive random testing*. in *Annual Asian Computing Science Conference*. 2004. Springer.
9. Chen, T.Y. *Fundamentals of test case selection: Diversity, diversity, diversity*. in *Software Engineering and Data Mining (SEDM), 2010 2nd International Conference on*. 2010. IEEE.
10. Barus, A.C., et al., *A cost-effective random testing method for programs with non-numeric inputs.* IEEE Transactions on Computers, 2016. **65**(12): p. 3509-3523.
11. Chan, K.P., T.Y. Chen, and D. Towey, *Restricted random testing: Adaptive random testing by exclusion.* International Journal of Software Engineering and Knowledge Engineering, 2006. **16**(04): p. 553-584.
12. Chen, T.Y., D.H. Huang, and Z.Q. Zhou. *Adaptive random testing through iterative partitioning*. in *International Conference on Reliable Software Technologies*. 2006. Springer.
13. Tappenden, A.F. and J. Miller, *A novel evolutionary approach for adaptive random testing.* IEEE Transactions on Reliability, 2009. **58**(4): p. 619-633.
14. Chan, K.P., T.Y. Chen, and D. Towey. *Forgetting test cases*. in *null*. 2006. IEEE.
15. Mao, C., T.Y. Chen, and F.-C. Kuo, *Out of sight, out of mind: a distance-aware forgetting strategy for adaptive random testing.* Science China Information Sciences, 2017. **60**(9): p. 092106.
16. Chen, T.Y., et al., *Mirror adaptive random testing.* Information and Software Technology, 2004. **46**(15): p. 1001-1010.
17. Huang, R., et al., *Enhancing mirror adaptive random testing through dynamic partitioning.* Information and Software Technology, 2015. **67**: p. 13-29.
18. Chen, T.Y., et al. *Adaptive random testing through dynamic partitioning*. in *Quality Software, 2004. QSIC 2004. Proceedings. Fourth International Conference on*. 2004. IEEE.
19. Mao, C., *Adaptive random testing based on two-point partitioning.* Informatica, 2012. **36**(3).
20. Chow, C., T.Y. Chen, and T. Tse. *The art of divide and conquer: an innovative approach to improving the efficiency of adaptive random testing*. in *Quality Software (QSIC), 2013 13th International Conference on*. 2013. IEEE.
21. Shahbazi, A., A.F. Tappenden, and J. Miller, *Centroidal voronoi tessellations-a new approach to random testing.* IEEE Transactions on Software Engineering, 2013. **39**(2): p. 163-183.

参考文献格式的问题很大