The Impact of Source Test Case Selection on the Effectiveness of Metamorphic Testing

Arlinta Christy Barus
Institut Teknologi Del
Indonesia
arlinta@del.ac.id

Tsong Yueh Chen, Fei-Ching Kuo
Swinburne University of Technology
Australia
{tychen, dkuo}@swin.edu.au

Huai Liu, Heinz W. Schmidt
RMIT University
Australia
{huai.liu,heinz.schmidt}@rmit.edu.au

ABSTRACT

Metamorphic Testing (MT) aims to alleviate the oracle problem. In MT, testers define metamorphic relations (MRs) which are used to generate new test cases (referred to as follow-up test cases) from the available test cases (referred to as source test cases). Both source and follow-up test cases are executed and their outputs are verified against the relevant MRs, of which any violation implies that the software under test is faulty. So far, the research on the effectiveness of MT has been focused on the selection of better MRs (that is, MRs that are more likely to be violated). In addition to MR selection, the source and follow-up test cases may also affect the effectiveness of MT. Since follow-up test cases are defined by the source test cases and MRs, selection of source test cases will then affect the effectiveness of MT. However, in existing MT studies, random testing is commonly adopted as the test case selection strategy for source test cases. This study aims to investigate the impact of source test cases on the effectiveness of MT. Since Adaptive Random Testing (ART) has been developed as an enhancement to Random Testing (RT), this study will focus on comparing the performance of RT and ART as source test case selection strategies on the effectiveness of MT. Experiment results show that ART outperforms RT on enhancing the effectiveness of MT.

Keywords

Metamorphic testing; source test case selection; adaptive random testing

1. INTRODUCTION

A test oracle is a mechanism to verify the correctness of computed outputs. However, situations exist where oracles may be unavailable or practically inapplicable. For example, a heuristic method does not guarantee to always deliver the most optimal solution. Hence, it is very difficult to verify the correctness of the output of a program implementing a heuristic [1]. This situation is known as an oracle problem. Metamorphic Testing (MT) is one of several testing strategies to alleviate the oracle problem[4]. MT uses some properties of software under test to define metamorphic relations (MRs). MRs are used to generate new test cases (referred to as follow-up test cases) from the existing test cases (referred to as source test cases). Next, the source and follow-up test case are executed and their outputs are then verified againts the corresponding MRs. The software under test can be considered faulty once the MRs are violated.

Obviously, the effectiveness of MT in revealing faults depends on the quality of MRs. There have been some studies to investigate the selection of good MRs [7, 23]. In addition to the quality of MRs, the effectiveness of MT should also depend on the source test cases. However, Random Testing (RT) is commonly used as the source test case selection strategy in MT. Since ART has been designed to improve the performance of Random Testing (RT) [5], we investigate the use of RT and ART as source test case selection strategies and their impact on the effectiveness of MT.

Section 2 of this paper explains some basic concepts of Metamorphic Testing and Adaptive Random Testing whereas Section 3 gives the motivation of this study. Detailed design of the experimental work is presented in Section 4. Section 5 reports the experiment results and their interpretation. Conclusion and future work are given in the last Section

MET'16, May 14-22, 2016, Austin, TX, USA
© 2016 ACM. ISBN 978-1-4503-4163-9/16/05...$15.00
DOI: http://dx.doi.org/10.1145/2896971.2896977

2. LITERATURE REVIEW

2.1 Metamorphic Testing

A test oracle is a mechanism that can be used to verify the correctness of computed outputs of a program [3]. We encounter a test oracle problem when (i) there is no such an oracle or (ii) the application of such an oracle becomes too expensive. To alleviate this problem, Chen et al. [4] developed the Metamorphic Testing (MT) approach which has been successfully applied in various application domains [6, 7, 8, 22]. The key idea of MT is the use of Metamorphic Relations (MRs) which are identified from the properties of the software under test. MRs are used to generate follow-up test cases from the source test cases. Both source and follow-up test cases are executed and their outputs are then verified against the corresponding MRs. Any violation of MRs implies that the software under test is faulty.

Let us use the search engine function to illustrate the idea of MT. Suppose P is a program implementing a search function f which searches websites containing any keywords specified. The test oracle for this program may be too expensive to apply because it is hard to verify whether P returns pages of all websites containing any keywords in the input. However, we know that if S_1 is an input and S_2 consists of S_1 and other keywords, then the returned pages (O_1) for S_1 ought to be a subset of the returned pages (O_2) for S_2. For example, a user would like to find some detective book titles using a certain search engine. He/she puts two authors' names Agatha Christie and Enid Blyton as keywords that is, $S_1 = \{$Agatha Christie, Enid Blyton$\}$. Then, he/she does a second search with one additional author Shidney Sheldon that is, $S_2 = \{$Agatha Christie, Enid Blyton, Shidney Sheldon$\}$. The returned pages for the second search (O_2) at least include all returned pages for the first search (O_1) or in other words, $O_1 \subset O_2$.

As program faults may be sensitive to different MRs, it is recommended to use more than one MR when applying MT. A main challenge of MT is to identify effective MRs [21].

Procedure MT Suppose the function f is implemented by a program P. The procedure of MT consists of the following steps:
1. Identify an MR for f .
2. Generate source test cases I_1 using an appropriate test case selection strategy.
2. Generate follow-up test cases I_2 from I_1 based on the MR.
2. Run P using I_1 and I_2 and get their outputs O_1 and O_2 correspondingly.
4. Verify I_1, I_2, O_1 and O_2 against the MR: if the MR does not hold, then P can be considered faulty.

The above procedure can be repeated for a group of MRs for the software under test.

2.2 Adaptive Random Testing

Adaptive Random Testing (ART) is a test case selection strategy to improve Random Testing (RT)[5]. As faults tend to cause erroneous behaviour to occur in contiguous regions of the input doman [11, 12], ART is based on the intuition that test cases close to each other are more likely to have similar failure behaviour than test cases further away from each other. Therefore ART attempts to select test cases widely spread across the input domain with the aim of finding failure with fewer number of test cases by than RT.

There are many ways to apply the principle of ART. One of them that has been widely used is Fixed-Size Candidate Set ART (FSCS-ART). Its first algorithm, which is for testing programs with numeric inputs, uses the Euclidean Distance to measure the distance between test cases [5]. It defines two groups of test cases: candidate set and executed set. The candidate test cases are selected randomly and then the candidate that is the most distant away from all executed test cases will be selected as the next test case. For each candidate, its distances to every executed test case are calculated and the minimum distant, say d_{min} is recorded. A candidate with the largest d_{min} will be selected as the next test case. This selection criterion is known as *max-min* criterion. If the selected test case does not reveal a failure, then it is added to the executed set. A new candidate set is selected again and then the above process of selecting the next test case is repeated. Testing stops whenever a failure is detected or testing resources are exhausted. The number of test cases required to reveal the first failure is known as *F-measure* [9]. This is used as the metric to measure the effectiveness of the technique.

Recently, ART has been used not only to test program with numeric inputs, but also to test programs with non-numeric inputs [20]. The category-choices technique [17] is adopted as the distance measure for non-numeric inputs. This technique is a specification-based method that identifies key input parameters or operational environment as categories and all possible disjoint groups of the values (namely choices) of the categories. The distance between two test cases is calculated by counting the number of different choices between them.

Kuo proposed some alternatives to *max-min* criterion of ART for testing programs with non-numeric inputs [10]. A proposed alternative is the *max-sum* criterion. Instead of considering the minimum values of the distances between each candidate and all executed test cases, the accumulated distances are recorded. The candidate having the largest accumulated distance is then selected as the next test case. This approach aims to alleviate the loosing of discriminatory power of *max-min* criterion when the number of executed test cases is large, because the number of categories for a given program is fixed and is normally not a large number. Kuo also considered only the n most recently executed test cases rather than all executed test cases. This approach is referred to as *aging* or *forgetting* approach which was first introduced by Chan et al. [25]. This approach aims to limit the number of distance comparison.

3. MOTIVATION

There have been many studies aiming to investigate how to improve the performance of Metamorphic Testing (MT). However, they have been focused on the identification of the Metamorphic Relations (MRs) that are more effective in revealing failures on the software under test. For the effectiveness of MRs, some factors such as the difference of execution paths between source and follow-up test cases [7, 23] and the strength of the MRs to reveal failures compared to existing test oracles [21] have been investigated.

In fact, in addition to the effectiveness of the MRs, the source and follow-up test cases also affect the performance of MT. Since the generation of follow-up test cases is dependant on the source test cases and the corresponding MRs, the selection of source test cases will affect the effectiveness of MT. However, in existing MT studies, random testing (RT) is commonly used as the test case selection strategy for source test cases. In this study, we aim to investigate the use of other source test case selection strategies and their impacts on the effectiveness of MT.

This study chooses Adaptive Random Testing (ART) as a new source test case selection strategy because ART has been developed as an enhancement to RT. ART is based on the intuition that two test cases close to each other are more likely to trigger the same failure behavior than those far from each other. Accordingly, ART aims to select test cases widely spread more than RT and hence is expected to reveal failure more effective than RT.

This study attempts to compare the use of RT and ART as source test case selection strategies for MT. We measure the effectiveness of MT using the F-measure. As many previous studies have demonstrated that ART has improved the effectiveness of RT, it is expected that the source test case selection using ART can enhance the performance of MT.

The present paper includes some results reported in Barus' work [2], in which `grep` is the only software under test. Here, we use more subject programs.

4. EMPIRICAL STUDY

4.1 Research question

We conducted an empirical study to answer the following research question: Can the use of ART in the source test case selection improve the effectiveness of MT?

4.2 Object programs: `grep` and SIEMENS Programs

4.2.1 `grep`

`grep` is a regular expression program of GNU that searches a given expression pattern in a given file [24]. It returns the matching lines in the input file. We chose `grep` as one of the object programs in this study as the released versions are freely accessed and `grep` has complex enough input file structure that are still feasible for the automated input generation purpose. Most faults in grep did not relate to the regular expression analyzer which is the part that we focus on this study. However, we found one real fault of grep fault program that is suitable for our use. As one fault is not sufficient, we generated 19 mutants using our own tool that applies two types of mutation operators, namely statement mutation and operator mutation.

4.2.2 SIEMENS Programs

SIEMENS programs have been widely used as the experimental subjects in software testing [16, 14, 2]. They are simple text pocessing utilities which were originally assembled by researchers at SIEMENS Corporate Research for software testing based on control-flow and data-flow test adequacy criteria [14]. We chose some of SIEMENS programs to be used in this study because of their manageable sizes and inputs, and their source codes, mutants, and test pools in SIR [15].

In this paper, we used five of the SIEMENS programs which are `printtokens`, `printtokens2`, `schedule`, `schedule2`, and `replace`. Program `printtokens` and `printtokens2` are lexical analysers. Both of them do exactly the same things and are based on the same specification but implemented differently and independently. These programs read an input file, and then split each line in the input file into tokens, identify token categories, and print out all the tokens and the categories in a specific order. Program `schedule` and `schedule2` perform priority scheduling. They receive a list of jobs and some commands of operations as inputs, and generate outputs of the ordered jobs based on their priorities. Like `printtokens` and `printtokens2`, these two programs also share similar basic specification; however `schedule` is non-preemptive while `schedule2` is preemptive. The last program, `replace`, is a command-line utility which takes three inputs: a search string, a replacement string, and an input file. It searches for occurrences of the search string in the input file, and produces an output file where each occurrence of the matched string is replaced with the replacement string. The search string is in a special format of regular expression and the replacement string is a text that can include some metacharacters.

4.3 Metamorphic Relations

In this study, we used metamorphic relations (MRs) for `grep` that have been defined in [2]. However, we only considered 6 out of 12 defined MRs because in the previous study [2], it was found that the remaining MRs were unable to reveal faults in any of the mutated versions. For the five SIEMENS programs, we used all MRs defined in [18] where each consisted of 3 MRs. Following is a list of representative MRs used in the study, one for each program (due to page limit, we are not able to list all MRs):

- `grep`
  Changing range character sets: The regular expression of follow-up test cases are generated from source test cases by re-arranging the elements in the range character set randomly (e.g. [1-3] is re-arranged into [231]). For this MR, the output of follow-up test cases must be equal to the output of the source test cases.

- `printtokens, printtokens2`
  Changing lower case into upper case: In this MR, the follow-up test cases are generated by changing all lower case characters in the source test input file to upper cases. This operation does not change the number of tokens in the output of the follow-up test cases. However output tokens with categories "keywords" are changed to categories "identifier" whilst other categories remain the same.

- `schedule, schedule2`
  Substituting the block and unblock commands: In this MR, firstly, the numbers of the command types of "blocked" and "unblocked" needs to be counted. Suppose the difference between these two numbers is n. Then, follow-up test cases are generated by deleting all of the commands having the smallest number of these two types, and add n commands of the other type in the input file. For example, if there are 5 commands "blocked" and 2 commands "unblock" in the input file of a source test case, then for the corresponding follow-up test case's input file, all commands "unblock" will be deleted and 3 commands "blocked" are added. Then, the number of printed job in output of follow-up test cases should be the same as that in the source test cases.

- `replace`
  Bracketing simple characters: In this MR, the search strings of the follow-up test cases are different to the search strings of the source test cases whereas the replacement strings and input files remain the same. The search string which is in a special format of regular expression may contain simple characters. In this MR, to form follow-up test cases, any simple characters in search strings of source test cases will be enclosed by a pair of square brackets. As the simple charactes still have similar meaning after being enclosed by a pair of square brackets (e.g. "a" is equivalent to "[a]") then

the output of follow-up test cases should be the same as output of the source test cases.

4.4 Variables and measures

4.4.1 Independent variable

The strategy to select source test cases of MT is the independent variable in this study. Three source test case selection strategies which are ART *max-min with aging*, ART *max-sum with aging* and RT, are included in this empirical study. For ART *max-min with aging*, ART *max-sum with aging*, the size of candidate set is 10. This size has been demonstrated as the maximum number showing the effectiveness of the FSCS-ART strategies [5]. For *aging* approach, prior relevant study [25] always arbitrarily defines the size of the executed test cases being considered. In this study, we propose a fix number which is 10 most recently executed test cases.

4.4.2 Dependent variable

To answer the research question above, the F-measure is used as a metric to evaluate the failure detection effectiveness of the three source test case selection strategies. This follows the relevant study in Barus' work [2] which used similar metric due to the nature of the comparison of RT and ART techniques. The F-measure is defined as the average number of test cases required to reveal the first failure[19]. A smaller F-measure reflects better failure detection performance. In addition, to better illustrate the performance improvement of ART over RT, F-ratio, referred to as the ratio of the F-measure of ART to that of RT, is normally used. If the value of F-ratio is less then 100%, it implies that ART outperforms RT in terms of using fewer test cases in detecting the first failure. Failures are identified whenever the relevant MRs are violated. For each fault, a thousand runs were performed to achieve a statistically significant confidence.

4.5 Categories and Choices

Categories and choices used for the object programs are taken from [20]. Due to the limited available documentation of the SIEMENS programs, they were derived based on the behaviour and source code of each program. For `grep`, the categories and choices were made based on the available user documentation, particularly on the part of regular expression analyzer. Details of the categories and choices for the object programs can be found in [20].

5. RESULTS

Table 1 - Table 6 give full results of F-ratio of both ART strategies for `grep` and the five SIEMENS programs. The values of "N/A" in the tables mean that the corresponding MR was not able to reveal any failures on relevant faulty version of the programs under test.

Table 7 presents direct pairwise comparisons of the F-measures of RT, ART *max-min with aging*, and ART *max-sum with aging* for all programs under test. Each cell in table denotes the number of faulty versions on which the test case selection strategies in the top row performing better than the strategies in the most left column. For example, for program `printtokens`, the cell in the most right of second row shows that ART *max-sum with aging* had a smaller F-measures than RT on all of 9 faults.

To test the significant level of the difference between the techniques under study, we conducted non-parametric test, a Friedman test. We did not use parametric test because the number of faults for each program was small and their F-measures were not normally distributed. We used the significance level α equal to 0.05 and used Holm-Benferroni method to evaluate the significant differences. **Bold** entries in the tables mean the significant differences of perfomance of corresponding compared tehniques. For example, for `replace`, ART *max-min with aging* outperformed RT significantly on 16 of the 23 faulty versions. However, ART *max-sum with aging* outperformed RT on 17 of the 23 faulty versions but not significantly.

The results show that ART *max-min with aging* outperformed RT significantly on `grep`, `schedule-2`, and `replace`. ART *max-sum with aging* outperformed RT significantly on 3 (`printtokens`, `printtokens-2`, and `schedule-2`) of the 5 SIEMENS programs. ART *max-sum with aging* outperformed ART *max-min with aging* significantly on `schedule-2`. On the other hand, RT could not significantly outperform either of the ART techniques on any program.

6. DISCUSSION AND CONCLUSION

From the results given in the previous section, we can see that the effectiveness of MT can be improved by generating source test cases using either ART *max-sum with aging* or ART *max-min with aging*. We also found that the use of ART *max-sum with aging* is slightly better than ART *max-min with aging* in the source test case generation of MT. In our study, F-measure was used to measure the effectiveness of test case selection strategies, due to the nature of the comparison of RT and ART. The F-measure results specifically showed that by using ART to select source test cases for MT, fewer test cases were required to detect the first failure than using RT.

Previous studies of MT merely investigate the improvement of MT by focussing on the quality of the MRs selected. However, this study has shown that the source test case selection could also give impact on the effectiveness of MT. In most previous studies, the source test case generation for MT was normally conducted based on RT. In this study, we used ART max-min with aging and ART max-sum with aging to select source test cases for MT. The experiment results have shown that the use of ART ART *max-min with aging* and ART *max-sum with aging* were also able to improve the effectiveness of MT.

7. ACKNOWLEDGEMENT

This research was supported by an Australia Research Council Grant (LP100200208)

8. REFERENCES

[1] A. C. Barus. *Testing Heuristic Method*, Software Engineering Techniques, Lecture Notes in Computer Science volume 4980, 2011, pp. 246-260.

[2] A. C. Barus. *An In-depth Study of Adaptive Random Testing for Testing Program with Complex Input Types*, Ph.D. dissertation, Swinburne University of Technology, 2010.

[3] E. J. Weyuker, *On testing non-testable programs*, The Computer Journal, vol. 25, No. 4, 1982, pp. 465-470.

Table 1: F-ratio of MT with ART *max-min with aging* and ART *max-sum with aging* for `grep`

version	ART *max-min with aging*						ART *max-sum with aging*					
	MR1	MR2	MR3	MR4	MR5	MR6	MR1	MR2	MR3	MR4	MR5	MR6
v1	N/A	39.08%	N/A	85.64%	270.74%	N/A	N/A	107.93%	N/A	149.78%	351.45%	N/A
v2	59.55%	N/A	N/A	N/A	57.22%	197.38%	108.64%	N/A	N/A	N/A	115.62%	197.09%
v3	N/A	N/A	N/A	79.97%	73.14%	N/A	N/A	N/A	N/A	154.68%	101.04%	N/A
v4	48.85%	122.36%	N/A	139.98%	54.08%	234.48%	108.15%	155.88%	N/A	156.52%	97.53%	218.02%
v5	70.29%	39.44%	N/A	73.27%	84.78%	241.97%	110.64%	107.19%	N/A	134.19%	91.30%	228.65%
v6	N/A	64.26%	N/A	66.78%	N/A	131.42%	N/A	128.08%	N/A	118.98%	N/A	133.20%
v7	54.83%	94.90%	N/A	82.13%	37.22%	103.35%	97.93%	91.32%	N/A	129.82%	97.68%	168.34%
v8	N/A	45.02%	N/A	N/A	N/A	93.74%	N/A	91.65%	N/A	N/A	N/A	100.10%
v9	N/A	N/A	N/A	76.68%	40.24%	55.50%	N/A	N/A	N/A	136.37%	100.09%	101.29%
v10	N/A	37.18%	N/A	71.05%	51.24%	N/A	N/A	103.12%	N/A	131.54%	95.04%	N/A
v11	62.38%	N/A	N/A	60.53%	76.03%	104.55%	112.69%	N/A	N/A	101.12%	97.86%	99.44%
v12	58.95%	N/A	N/A	57.96%	37.26%	92.76%	92.40%	N/A	N/A	102.28%	103.36%	88.47%
v13	64.33%	N/A	N/A	N/A	N/A	97.73%	114.87%	N/A	N/A	N/A	N/A	167.67%
v14	N/A	55.26%	N/A	58.90%	28.57%	77.88%	N/A	110.16%	N/A	121.75%	28.57%	84.88%
v15	56.09%	N/A	N/A	59.28%	25.06%	77.42%	93.65%	N/A	N/A	103.24%	71.80%	64.63%
v16	111.04%	105.19%	127.88%	58.63%	84.92%	91.38%	103.42%	141.32%	127.53%	88.03%	100.77%	98.99%
v17	59.97%	N/A	N/A	N/A	62.54%	N/A	96.81%	N/A	N/A	N/A	90.86%	N/A
v18	66.34%	42.09%	N/A	63.47%	37.80%	N/A	109.21%	97.49%	N/A	115.74%	93.13%	N/A
v19	N/A	N/A	N/A	51.76%	50.70%	91.19%	N/A	N/A	N/A	108.85%	87.19%	94.48%
v20	57.24%	N/A	N/A	59.65%	N/A	N/A	76.94%	N/A	N/A	86.22%	N/A	N/A

Table 2: F-ratio of MT with ART *max-min with aging* and ART *max-sum with aging* for `printtokens`

version	ART *max-min with aging*			ART *max-sum with aging*		
	MR1	MR2	MR3	MR1	MR2	MR3
v1	N/A	N/A	50.10%	N/A	N/A	45.04%
v2	N/A	N/A	51.40%	N/A	N/A	45.94%
v3	N/A	N/A	50.12%	N/A	N/A	44.94%
v4	N/A	N/A	50.06%	N/A	N/A	45.10%
v5	N/A	13.74%	58.95%	N/A	12.03%	51.46%
v6	51.40%	N/A	50.33%	46.17%	N/A	45.14%
v7	N/A	N/A	50.31%	N/A	N/A	45.30%

Table 3: F-ratio of MT with ART *max-min with aging* and ART *max-sum with aging* for `printtokens-2`

version	ART *max-min with aging*			ART *max-sum with aging*		
	MR1	MR2	MR3	MR1	MR2	MR3
v1	116.22%	131.52%	89.63%	92.20%	159.54%	72.13%
v2	135.73%	128.45%	89.18%	124.58%	145.38%	71.54%
v3	53.44%	71.10%	89.14%	35.96%	88.24%	71.62%
v4	25.68%	96.55%	85.23%	12.36%	110.38%	71.72%
v5	35.81%	68.23%	89.40%	21.79%	87.96%	72.25%
v6	101.63%	67.56%	89.39%	112.32%	86.37%	71.50%
v7	119.49%	107.36%	86.07%	105.86%	120.16%	69.86%
v8	42.36%	69.60%	89.46%	34.71%	87.41%	71.43%
v9	N/A	106.47%	90.54%	N/A	148.19%	73.40%
v10	N/A	68.87%	75.47%	N/A	86.57%	68.24%

Table 4: F-ratio of MT with ART *max-min with aging* and ART *max-sum with aging* for `schedule`

version	ART *max-min with aging*			ART *max-sum with aging*		
	MR1	MR2	MR3	MR1	MR2	MR3
v1	41.82%	N/A	101.34%	39.40%	N/A	154.16%
v2	N/A	N/A	100.86%	N/A	N/A	156.83%
v3	N/A	N/A	103.10%	N/A	N/A	153.70%
v4	N/A	N/A	103.78%	N/A	N/A	154.29%
v5	N/A	N/A	28.48%	N/A	N/A	41.81%
v6	42.06%	N/A	102.66%	39.74%	N/A	156.53%
v7	N/A	N/A	103.18%	N/A	N/A	156.74%
v8	N/A	N/A	106.29%	N/A	N/A	155.68%
v9	N/A	N/A	N/A	N/A	N/A	N/A

Table 5: F-ratio of MT with ART *max-min with aging* and ART *max-sum with aging* for `schedule-2`

version	ART *max-min with aging*			ART *max-sum with aging*		
	MR1	MR2	MR3	MR1	MR2	MR3
v1	43.07%	134.21%	87.52%	24.40%	70.89%	49.29%
v2	59.17%	N/A	52.48%	36.09%	N/A	57.23%
v3	42.62%	N/A	36.98%	24.67%	N/A	27.15%
v4	43.72%	N/A	37.70%	24.20%	N/A	26.66%
v5	43.35%	N/A	52.72%	24.24%	N/A	36.43%
v6	42.74%	N/A	37.85%	24.38%	N/A	26.44%
v7	59.03%	N/A	54.35%	36.26%	N/A	57.27%
v8	N/A	N/A	37.74%	N/A	N/A	26.84%
v9	43.92%	N/A	37.62%	24.50%	N/A	26.51%
v10	42.95%	N/A	37.89%	24.34%	N/A	26.58%

Table 6: F-ratio of MT with ART *max-min with aging* and ART *max-sum with aging* for `replace`

version	ART *max-min with aging*			ART *max-sum with aging*		
	MR1	MR2	MR3	MR1	MR2	MR3
v1	N/A	76.76%	81.87%	N/A	76.76%	139.80%
v2	N/A	N/A	33.71%	N/A	N/A	45.58%
v3	N/A	N/A	32.07%	N/A	N/A	56.59%
v4	N/A	N/A	121.46%	N/A	N/A	164.10%
v5	170.25%	N/A	114.18%	312.84%	N/A	67.76%
v6	N/A	N/A	62.86%	N/A	N/A	46.17%
v7	N/A	N/A	71.84%	N/A	N/A	70.63%
v8	154.83%	N/A	31.86%	174.43%	N/A	34.92%
v9	N/A	N/A	54.03%	N/A	N/A	52.45%
v10	N/A	N/A	49.53%	N/A	N/A	56.35%
v11	N/A	N/A	34.87%	N/A	N/A	44.22%
v12	60.56%	58.84%	184.05%	47.12%	58.84%	104.09%
v13	N/A	103.68%	N/A	N/A	103.68%	N/A
v14	N/A	N/A	N/A	N/A	N/A	N/A
v15	N/A	N/A	N/A	N/A	N/A	N/A
v16	N/A	N/A	N/A	N/A	N/A	N/A
v17	N/A	N/A	N/A	N/A	N/A	N/A
v18	N/A	N/A	N/A	N/A	N/A	N/A
v19	N/A	N/A	N/A	N/A	N/A	N/A
v20	N/A	N/A	N/A	N/A	N/A	N/A
v21	N/A	N/A	N/A	N/A	N/A	N/A
v22	N/A	N/A	N/A	N/A	N/A	N/A
v23	N/A	N/A	N/A	N/A	N/A	N/A
v24	N/A	N/A	N/A	N/A	N/A	N/A
v25	N/A	N/A	N/A	N/A	N/A	N/A
v26	N/A	N/A	N/A	N/A	N/A	N/A
v27	99.37%	N/A	N/A	101.63%	N/A	N/A
v28	N/A	56.38%	35.23%	N/A	56.38%	43.47%
v29	N/A	N/A	N/A	N/A	N/A	N/A
v30	N/A	56.81%	35.31%	N/A	56.81%	43.50%
v31	N/A	N/A	N/A	N/A	N/A	N/A
v32	N/A	N/A	N/A	N/A	N/A	N/A

[4] T. Y. Chen, S. Cheung, and S. Yiu, *Metamorphic testing : a new approach for generating next test cases*, Department of Computer Science , Hong Kong University of Science and Technology, Hong Kong, Tech. Rep. HKUST-CS98-01, 1998.

[5] T. Y. Chen, H. Leung, and I. K. Mak, 2004. *Adaptive Random Testing*, In Proceedings of the 9th Asian Computing Science Conference, ser. Lecture Notes in Computer Science, vol. 3321, 2004, pp. 320-329.

[6] W. K. Chan, S. C. Cheung, and K. R. P. H. Leung. *A Metamorphic Testing Approach for Online Testing of Service-Oriented Software Applications*, a Special Issue on Service Engineering of International Journal of Web Services Research, vol. 4, No. 2, 2007, pp. 60-80.

[7] T. Y. Chen, D. H. Huang, T. H. Tse, and Z. Q. Zhou. *Case studies on the selection of useful relations in metamorphic testing*, in Proceedings of the 4th lbero-American Symposium on Software Engineering and Knowledge Engineer-ing (JIISIC). Polytechnic University of Madrid, 2004, pp. 569-583.

[8] T. Y. Chen, T. H. Tse, and Z. Q. Zhou. *Semi-proving: an integrated method based on global symbolic evaluation and metamorphic testing*, in Proceedings of the ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA). ACM Press, New York, 2002, pp. 191-195.

[9] T. Y. Chen, T. H. Tse, and Y. T. Yu, *Proportional sampling strategy: A compendium and some insights*, Information and Software Technology, vol. 58, No. 1, 2001, pp. 65-81.

[10] F.-C. Kuo, *On adaptive random testing*, Ph.D. dissertation, Swinburne University of Technology, 2006.

[11] P. E. Ammann and J. C. Knight, *Data diversity: an approach to software fault tolerance*, IEEE Transactions on Computers, vol. 37, No. 4, 1998, pp. 418-425.

[12] P. G. Bishop, *The variation of software survival times for diffrent operational input profiles*, in FTSC-23. Digitest of Papers, the Twenty-Third International Symposium on Fault-Tolerant Computing. IEEE Computer Society Press, 1993, pp. 98-107.

[13] R. Merkel, *Analysis and enhancements of adaptive random testing*, Ph.D. dissertation, Swinburne University of Technology, 2005.

[14] M. Hutchins, H. Foster, T. Goradia, and T. Ostrand, *Experiments on the effectiveness of dataflow and controlfow-based test adequacy criteria*, in Proceedings of International Conference of Software Engineering, 2004, pp. 191-200.

Table 7: Pairwise Comparison of the F-measures of Different Selection Strategies

Program	Technique	RT	ART *max-min with aging*	ART *max-sum with aging*
`grep`	RT	N/A	**57**	26
	ART *max-min with aging*	**12**	N/A	11
	ART *max-sum with aging*	43	58	N/A
`printtokens`	RT	N/A	9	9
	ART *max-min with aging*	0	N/A	**9**
	ART *max-sum with aging*	0	**0**	N/A
`printtokens-2`	RT	N/A	20	**20**
	ART *max-min with aging*	8	N/A	17
	ART *max-sum with aging*	**8**	11	N/A
`schedule`	RT	N/A	3	3
	ART *max-min with aging*	7	N/A	2
	ART *max-sum with aging*	7	8	N/A
`schedule-2`	RT	N/A	**19**	**20**
	ART *max-min with aging*	**1**	N/A	**18**
	ART *max-sum with aging*	**0**	**2**	N/A
`replace`	RT	N/A	**16**	17
	ART *max-min with aging*	**7**	N/A	10
	ART *max-sum with aging*	6	13	N/A

[15] G. Rothermel, E. Sebastian, H. Do, and A. Kinneer, *Software-artifact infrastructure repository*, [Online]. Available: http://sir.unl.edu.

[16] S. G. Elbaum, A. G. Malishevsky, and G. Rothermel, *Prioritizing test cases for regression testing*, in In Proceedings of the 2000 ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA 2000), ACM SIGSOFT Software Engineering Notes, 2000, pp. 102-112.

[17] T. J. Ostrand and M. J. Balcer, *The category-partition method for specifying and generating functional tests*, Communications of the ACM, vol. 31, No. 6, June 1988.

[18] X. Xie, W. E. Wong, T. Y. Chen, B. W. Xu, *Metamorphic Slice: An Application in Spectrum-based Fault Localization*, Information and Software Technology, 2013, vol.55, No. 5, pp. 866-879.

[19] T. Y. Chen, F.-C. Kuo, R. Merkel, *On the statistical properties of testing effectivenes measures*, Journal of Systems and Software, vol. 79, No. 5, 2006, pp. 591-601.

[20] A. C. Barus, T. Y. Chen, F.-C. Kuo, H. Liu, R. Merkel, G. Rothermel, *A Novel Linear-Order Algorithm for Adaptive Random Testing of Programs with Non-Numeric Inputs*, Technical Report TR-UNL-CSE-2014-0004, University of Nebraska Lincoln,US, 2014.

[21] H. Liu, F.-C.Kuo, D. Towey, T. Y. Chen, *How Effectively Does Metamorphic Testing Alleviate the Oracle Problem?*, IEEE Transactions on Software Engineering, vol. 40, No. 1, January 2014.

[22] Z. Q. Zhou, S. Zhang, M. Hagenbuchner, T. H. Tse, F. -C. Kuo, T. Y. Chen, *Automated functional testing of online search services*, Software Testing, Verification and Reliability, vol. 22, No. 4, 2012, pp. 221-243.

[23] Y.Cao, Z. Q. Zhou, T. Y. Chen, *On the Correlation between the Effectiveness of Metamorphic Relations and Dissimilarities of Test Case Executions*, in Proceedings of the 13th International Conference of Quality Software, 2013, pp. 153-162.

[24] The GNU Project, *Grep home page*, accessed 2015-11-01 [Online], Available: http://www.gnu.org/software/grep/manual/grep.html.

[25] K.-P. Chan, T. Y. Chen, and D. Towey, *Forgetting test cases*, In Proceedings of the 30th Annual International Computer Software and Applications Conference, COMPSAC 06, 2006, pp 485-494.