Does adaptive random testing deliver a higher confidence than random testing?


Published Version: https://doi.org/10.1109/QSIC.2008.23

Repository homepage: https://researchrepository.rmit.edu.au

© 2008 IEEE.
Thank you for downloading this document from the RMIT Research Repository.

The RMIT Research Repository is an open access database showcasing the research outputs of RMIT University researchers.


Citation:

See this record in the RMIT Research Repository at:

Version: Accepted Manuscript

Copyright Statement: © 2008 IEEE.

Link to Published Version:
http://dx.doi.org/10.1109/QSIC.2008.23
Abstract

Random testing (RT) is a fundamental software testing technique. Motivated by the rationale that neighbouring test cases tend to cause similar execution behaviours, adaptive random testing (ART) was proposed as an enhancement of RT, which enforces random test cases evenly spread over the input domain. ART has always been compared with RT from the perspective of the failure-detection capability. Previous studies have shown that ART can use fewer test cases to detect the first software failure than RT. In this paper, we aim to compare ART and RT from the perspective of program-based coverage. Our experimental results show that given the same number of test cases, ART normally has a higher percentage of coverage than RT. In conclusion, ART outperforms RT not only in terms of the failure-detection capability, but also in terms of the thoroughness of program-based coverage. Therefore, ART delivers a higher confidence of the software under test than RT even when no failure has been revealed.

Keywords: Failure-Based Testing, Random Testing, Adaptive Random Testing, Test Coverage Criteria.

1. Introduction

Software quality has become one of the most essential concerns in the software development process. Software testing is a popular software engineering approach to guaranteeing the software quality. Many testing methods have been proposed to selectively choose program inputs for testing (namely, test cases), with the purpose of effectively detecting software failures. Random testing (RT) is a basic software testing technique, which simply generates test cases at random from all possible program inputs (namely, input domain) [17, 23]. RT technique can be used for both reliability estimation [15, 30] and debug testing [14, 26, 28, 32, 37].

Some researchers [1, 2, 13, 33] have independently observed that program inputs that cause software failures (namely, failure-causing inputs) are commonly clustered into contiguous regions. Under such a common situation, it can be argued that if a test case does not reveal any failure, it is very unlikely for its neighbours to reveal failures. Based on this intuition, adaptive random testing (ART) [7] has been proposed to enhance the failure-detection capability of RT. In ART, test cases are not only randomly selected, but also evenly spread over the whole input domain. A number of ART algorithms have been developed, such as fixed-size-candidate-set ART [7], restricted random testing [3], quasi-random testing [9], and lattice-based ART [21]. Experimental studies have been conducted on these algorithms, and it has been shown that ART normally uses fewer test cases to detect the first software failure than pure RT. A recent theoretical study [8] has shown that the performance of ART is very close to that of the optimal testing strategy that could be developed without using any information about failure locations.

Intuitively speaking, adjacent test cases are very likely to cause the program under test to execute similarly, and thus to exhibit similar execution behaviours. As ART enforces its test cases spread away from one another, it is intuitively expected that ART is more likely to trigger more distinct execution behaviours than RT, and hence to cause the program under test to exhibit failures more quickly. However, if two program inputs are spread away from each other, it does not necessarily imply that these two inputs should execute distinct statements or branches of the program under test. In other words, ART may not necessarily achieve a higher coverage than RT. Although there have been many studies on the failure-detection capability of ART, no investigation has been conducted on how thoroughly the program under test will be covered by test cases selected by
ART; needless to say, there was no study on the comparison between the coverage achieved by ART and RT.

Test coverage criteria have been considered in many testing techniques to guide the test case selection process such that the program under test can be thoroughly covered. Control-flow and data-flow coverage criteria are two typical examples of test coverage criteria. Control-flow criteria [23] concern whether test cases will fully exercise certain control constructs of the program under test, while data-flow coverage criteria [20, 25] concern whether test cases will completely execute certain patterns of data manipulation in the program. Previous studies [18, 34, 35, 36] have shown that the testing effectiveness is strongly correlated to the coverage achieved by a test set (that is, a set of test cases). Besides selecting test cases, these coverage criteria can also be applied to measure the adequacy of test cases [38]. In this paper, we will conduct a series of experiments to evaluate and compare both control-flow and data-flow coverage attained by ART algorithms and pure RT.

The paper is organised as follows. In Section 2, we introduce some background information about ART and test coverage criteria. In Section 3, we report our experiments and the related results. In Section 4, we conclude the paper.

2. Preliminaries

2.1. ART algorithms

In this paper, we work on two particular ART algorithms, namely fixed-size-candidate-set ART (FSCS-ART) [7] and an enhancement of FSCS-ART, namely FSCS-ART with partitioning by edge and centre (ECP-FSCS-ART) [5]. These ART algorithms and their failure-detection capabilities are described as follows.

In FSCS-ART [7], there are two sets of test cases. One set (namely the executed set) contains all executed test cases, denoted by $E = \{e_1, e_2, \ldots, e_n\}$, where $n$ is the number of executed test cases. The other set (namely the candidate set) contains $k$ randomly generated inputs, denoted by $C = \{c_1, c_2, \ldots, c_k\}$, where $k$ is fixed throughout the testing process. An element of $C$ (namely candidate) will be selected as the next test case if it has the longest distance to its nearest neighbour in $E$. The “distance” in FSCS-ART usually refers to Euclidean distance if the program under test only has numeric inputs. Some investigations on the measurement of distances among non-numeric inputs can be found in [10, 11, 19, 22]. Figure 1.a illustrates FSCS-ART in a two-dimensional space. In Figure 1.a, there are one executed test case ($e_1$) and three random candidates ($c_1, c_2, \text{and} \ c_3$). $c_2$ is the farthest candidate from $e_1$, and will be selected as the next test case. Readers can refer to [7] for details of FSCS-ART algorithm.

F-measure, which refers to the expected number of test cases needed to reveal the first software failure, is a commonly used metric for the failure-detection capability of ART. All previous studies used F-measure to evaluate and compare the effectiveness of ART and RT (the appropriateness of F-measure in the study of ART has been justified in [8]). Empirical studies conducted in [7] have shown that FSCS-ART generally has a smaller F-measure (that is, a better failure-detection capability) than RT. However, some recent studies [6] have shown that the improvement on the failure-detection capability of FSCS-ART over RT diminishes with the increase of the dimension of input domain (that is, the number of input parameters of the program under test). Moreover, FSCS-ART may have a larger F-measure than RT under some situations.

Chen et al. [6] have observed that FSCS-ART prefers to select test cases from the edge part of the input domain rather than from the centre part, and such a preference, namely the edge preference, becomes more significant when the dimension of input domain is higher. They further pointed out that the edge preference is a cause of the deterioration of the failure-detection capability of FSCS-ART for high dimensional cases. In order to offset the edge preference of the original FSCS-ART, a new algorithm ECP-FSCS-ART [5] has been proposed. ECP-FSCS-ART works as follows. Suppose that there are $n$ executed test cases. For the selection of the $(n+1)$th test case, ECP-FSCS-ART first divides the whole input domain into $n+1$ equal-sized disjoint partitions from the edge to the centre of the input domain (that is, the partitioning scheme in ECP-FSCS-ART is dynamic along the testing process). Random candidates are then generated only from the partitions to which no executed test case belongs (note that the number of partitions is always larger than the number of executed test cases). The next test case will be selected from these candidates (the criterion identifying the best candidate in ECP-FSCS-ART is the same as that in FSCS-ART). In this way, different partitions will have similar densities of test cases, the edge preference will be well offset, and thus test cases will be spread more evenly. The basic procedure of ECP-FSCS-ART is illustrated in Figure 1.b. In this figure, the two-dimensional input domain is divided into two equal-sized disjoint partitions $D_1$ and $D_2$, where $D_1$ is located in the centre of the input domain, and $D_2$ is right outside $D_1$. An executed test case $e_1$ happens to lie inside $D_2$, and three candidates $c_1, c_2$ and $c_3$ are randomly generated inside the input domain. However, $c_2$ is in $D_2$, where there is al-
ready an executed test case $e_1$. Hence, ECP-FSCS-ART generates a new candidate $c'_2$ instead of $c_2$ to ensure that all candidates are from $D_1$, where there is no executed test case (that is, where the density of test cases is low). $c'_2$ will finally be selected as the next test case. The detailed algorithm of ECP-FSCS-ART can be found in [5]. Some simulation studies in [5] have shown that ECP-FSCS-ART significantly improves the failure-detection capability of FSCS-ART, especially for the cases of high dimension.

Control-flow coverage criteria [23] consider some control constructs of the program under test. For example, in statement testing strategy [23], test cases are selected such that all statements in the program are executed at least once. Condition coverage [23] is another example of control-flow testing strategy, which is more general and implies statement testing. In condition coverage testing, test cases are required to “ensure that each condition in a decision takes on all possible outcomes at least once”. Readers may be interested to find more control-flow coverage criteria in [23].

Figure 1. Illustrations of FSCS-ART and ECP-FSCS-ART and RT in a two-dimensional space

Briefly speaking, both ART algorithms normally have better failure-detection capabilities than RT, but they are different in terms of the failure-detection capability as well as the evenness of test case distribution, especially when the dimension of input domain is high.

2.2. Test coverage criteria

There are various test coverage criteria in the literature [38]. Some coverage criteria, namely specification-based criteria, require the selected test cases to thoroughly execute certain features of the software specification. Different from specification-based criteria, program-based coverage criteria require the selected test cases to thoroughly exercise the program under test from some perspectives. This paper will focus on program-based criteria, which can be further categorised into control-flow and data-flow criteria, as presented in the following.

According to data-flow coverage criteria [20, 25], test cases should thoroughly exercise certain patterns of data manipulation within the program under test. Patterns of data manipulation include definition of a datum (abbreviated as def), where a value is allocated to the datum; and usage of a datum (abbreviated as use), where the datum’s value is read by an operation. In addition, use can be further classified into c-use, where a datum is used in a computational expression or as an output; and p-use, where a datum appears in a predicate within the program. Many data-flow testing strategies have been proposed based on the above concepts, such as all-defs, all-p-uses, all-c-uses and all-du-pairs coverage.

Many previous studies have shown the advantages of test coverage criteria from various perspectives. Hutchins et al. [18], for example, have used some coverage testing techniques to detect failures in some real-life programs. It was observed that the higher coverage a test set could achieve, the more failures would be revealed. Wong et al. pointed out that the coverage achieved by a test set is strongly correlated to the failure-detection capability [34], while the size reduction of a test set almost has no impact on the failure-detection capability as long as the coverage remains unchanged [35, 36]. Rothermel et al. [27] have used some coverage criteria as test case prioritisation schemes. It was found that test coverage criteria can significantly enhance the failure-detection capability of the test case prioritisation technique. Chen et al. [4] also observed that the coverage technique can even be applied to improve the effectiveness of software reliability estimation.

Test coverage criteria are very useful in the measure-
ment of the quality of a test set [38]. Intuitively speaking, if a test set covers the program under test more thoroughly in terms of a certain coverage criterion $\epsilon$, this set can then be regarded as more adequate and thus more qualified with respect to $\epsilon$. Furthermore, if a testing strategy can generate a test set with a higher percentage of coverage (referred to as coverage percentage hereafter) with respect to $\epsilon$, we say that this testing strategy is a better method according to $\epsilon$.

3. Comparing coverage achieved by adaptive random testing and random testing

3.1. Experimental settings

In this study, we use the xSuds tool developed by Telcordia [29] to analyse the program-based coverage of a testing method. xSuds can evaluate both the control-flow and data-flow coverage. For control-flow coverage, xSuds evaluates “block coverage” and “decision coverage”. A block refers to a sequence of statements, whose execution will not be interrupted by any decision. Therefore, the block coverage in xSuds is effectively equivalent to statement coverage [23]. The “decision coverage” in xSuds requires test cases to exercise both the true and false outcomes of all conditions in each decision of the program under test, that is, the “decision coverage” in xSuds is effectively equivalent to condition coverage in [23]. xSuds also measures two data-flow coverage, namely “c-uses coverage” and “p-uses coverage”. In this paper, we will measure all these four coverage percentages for FSCS-ART, ECP-FSCS-ART and pure RT.

As mentioned before, ART algorithms have different failure-detection capabilities for different dimensions of the input domain. In order to find whether the coverage percentages of these two ART algorithms also depend on the dimension of the input domain, we selected five programs with different dimensions of input domains for this study. Among these programs, airy and bessjy, which compute two special functions (airy and ordinary bessel functions, respectively), are extracted from Numerical Recipes [24]. Programs quadratic and cubic, which calculate complex roots of quadratic and cubic equations, respectively, can be found in GNU Scientific Library [16]. The fifth program, namely tcas, is an on-board aircraft conflict detection and resolution system [18], which can be obtained from Software-Artifact Infrastructure Repository [12]. The details of these five programs are listed in Table 1. In the table, the input domains of these programs are defined according to program specifications. It should be noted that in tcas, there are some input parameters of Boolean type, such as the 2nd, 3rd, 10th, 11th and 12th parameters. The true and false values of these Boolean parameters are traditionally referred to as 1 and 0 in integral format, respectively, as shown in Table 1. Table 1 also reports the total numbers of blocks, decisions, c-uses and p-uses of all subject programs, which are obtained by the xSuds tool.

We evaluate the coverage percentages of FSCS-ART, ECP-FSCS-ART and RT through the following procedure.

1. Select a subject program and a test case selection strategy.

2. Generate a test set, with the number of test cases (that is, the size of the test set) being 1, 2, · · · , 10, 20, · · · , 100, 200, · · · , 1000.

3. Use xSuds to evaluate the block, decision, c-uses, and p-uses coverage percentages of the test set generated in Step 2.

4. Repeat Steps 2 and 3 for a sufficient number (S) of times to achieve certain confidence level $(1 - \alpha) \times 100\%$ and accuracy range $\pm r\%$. According to the central limit theorem [31], we can get

$$S = \left(100 \cdot \Phi^{-1}(\frac{\alpha}{2}) \cdot \sigma \right)^2,$$

where $\mu$ and $\sigma$ are the mean value and the standard deviation of coverage percentages collected in Step 3, respectively, and $\Phi^{-1}(\cdot)$ denotes the inverse standard normal distribution function. In this paper, we set the confidence level and the accuracy range as 95% ($\alpha = 0.05$) and $\pm 5\%$ ($r = 5$), respectively.

3.2. Experimental results

The experimental results are plotted in Figures 2 to 6. We only plot the block and p-uses coverage percentages. The experimental results of decision and c-uses coverage are similar to those shown in Figures 2 to 6. In these figures, the x-axis represents the number of test cases in the logarithmic scale, and y-axis denotes the average coverage percentage attained by various testing strategies.

Based on the experimental data, we have the following observations.

- For program airy (Figure 2), FSCS-ART and ECP-FSCS-ART have similar coverage percentages, but both of them have higher coverage percentages than RT.
- For program bessjy (Figure 3),
Table 1. The basic information of subject programs

<table>
<thead>
<tr>
<th>program</th>
<th>dimension</th>
<th>input domain</th>
<th>number of blocks</th>
<th>number of decisions</th>
<th>number of c-uses</th>
<th>number of p-uses</th>
</tr>
</thead>
<tbody>
<tr>
<td>airy</td>
<td>1</td>
<td>(-2000, 0)</td>
<td>217</td>
<td>100</td>
<td>412</td>
<td>174</td>
</tr>
<tr>
<td>bessy</td>
<td>2</td>
<td>(-100, -100)</td>
<td>133</td>
<td>62</td>
<td>239</td>
<td>116</td>
</tr>
<tr>
<td>cubic</td>
<td>4</td>
<td>(-1000, -1000, -1000, -1000)</td>
<td>133</td>
<td>62</td>
<td>239</td>
<td>116</td>
</tr>
<tr>
<td>tcas</td>
<td>12</td>
<td>(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)</td>
<td>133</td>
<td>62</td>
<td>239</td>
<td>116</td>
</tr>
</tbody>
</table>

- ECP-FSCS-ART always has higher coverage percentages than RT.
- FSCS-ART has marginally higher coverage percentages than RT. When the number of test cases is small (less than 50 test cases), FSCS-ART has higher coverage percentages than RT. However, when the test set has a larger size (greater than 50 test cases), FSCS-ART and RT have similar coverage percentages.

For program quadratic (Figure 4),

- There is no significant difference between the coverage percentages of FSCS-ART and RT.
- When the number of test cases is large (greater than 30 test cases), ECP-FSCS-ART has higher coverage percentages than FSCS-ART and RT.

For program cubic (Figure 5),

- When the size of the test set is very small (less than 5 test cases), RT has marginally higher coverage percentages than both ART algorithms.
- When the size of the test set is very large (greater than 400 test cases), both FSCS-ART and ECP-FSCS-ART have higher coverage percentages than RT.
- Under other situations, the coverage percentages of ECP-FSCS-ART are higher than those of RT and FSCS-ART.

For program tcas (Figure 6),

- When the number of test cases is large (greater than 300 test cases), all three testing strategies have similar coverage percentages.
- FSCS-ART has lower coverage percentages than RT when the size of the test set is small (less than 20 test cases).

- ECP-FSCS-ART has higher coverage percentages than RT when the size of the test set is not too large (up to 300 test cases).

For ease of comparison, we also summarise the experimental results in Table 2, which shows the ratios between the coverage percentages of ART and RT. The maximal, minimal and average values of these ratios are given in the table. There are totally 20 cases (that is, 5(programs) × 4(coverage criteria)). On average, FSCS-ART has higher coverage percentages than RT in 16 cases, while ECP-FSCS-ART is better than RT in all 20 cases. In other words, the improvement of coverage percentages of ECP-FSCS-ART over RT is more consistent than that of FSCS-ART. Further comparing two ART algorithms, it can be observed that ECP-FSCS-ART has more thorough coverage than FSCS-ART in 18 out of 20 cases. Briefly speaking, both FSCS-ART and ECP-FSCS-ART can cover the programs under test more thoroughly than pure RT, and ECP-FSCS-ART has higher coverage percentages than FSCS-ART.

In general, the experimental data are consistent with the intuition. For the same number of test cases, ECP-FSCS-ART normally has the highest coverage percentages, followed by FSCS-ART and RT in descending order. When considering the impact of the input domain’s dimension, we observed that the difference between ART algorithms and RT diminishes with the increase of dimension, but the coverage of ECP-FSCS-ART is apparently more thorough than that of FSCS-ART for high dimensional cases.

Based on the above results and previous studies, we can find that the failure-detection capabilities of ART/RT techniques are correlated to the coverage percentages. As shown in [5], ECP-FSCS-ART generally has the best failure-detection capability, followed by FSCS-ART and RT in descending order. The same ranking can also be observed when considering the coverage percentages of these three testing techniques. Such a correlation between the failure-detection capability and the coverage percentage is not surprising. It is generally believed that a test set with higher coverage...
Figure 2. Coverage percentages of FSCS-ART, ECP-FSCS-ART and RT on the *airy* program.

Figure 3. Coverage percentages of FSCS-ART, ECP-FSCS-ART and RT on the *bessjy* program.

Figure 4. Coverage percentages of FSCS-ART, ECP-FSCS-ART and RT on the *quadratic* program.
is more likely to have a better failure-detection capability than a test set with lower coverage [18, 34]. Since ART generally covers the program under test more thoroughly than RT, it is intuitive for the former to be more effective than the latter.

Previous studies [6] have shown that the failure-detection capability of an ART algorithm depends on how evenly the algorithm spreads its test cases. However, intuitively speaking, evenly spreading test cases is not necessarily related to coverage, because spreading two program inputs away from each other in the input domain does not necessarily imply that these inputs should execute distinct statements of the program under test. Nevertheless, our results show that ART does have a higher coverage than RT. ART was designed specifically to enhance the failure-detection capability of RT by revealing software failures more quickly, not for the purpose of achieving a higher coverage. Therefore, it is not a great surprise that the improvement of ART on the failure-detection capability is more significant than the enhancement on the coverage percentages.

In summary, ART not only brings a better failure-detection capability, but also has higher coverage percentages than RT. As a consequence, ART delivers a higher confidence of the software than RT even when no failure is detected.

4. Conclusion

Adaptive random testing (ART) is an enhancement of random testing (RT). The basic principle of ART is to evenly spread random test cases over the input domain. All previous studies of ART used the failure-detection capability to evaluate the effectiveness of ART. It has
Table 2. Ratios between coverage percentages of FSCS-ART/ECP-FSCS-ART and RT

2.a Coverage ratios on airy

<table>
<thead>
<tr>
<th></th>
<th>FSCS-ART vs. RT</th>
<th>ECP-FSCS-ART vs. RT</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>max</td>
<td>min</td>
</tr>
<tr>
<td>block coverage</td>
<td>1.1604</td>
<td>1.0000</td>
</tr>
<tr>
<td>decision coverage</td>
<td>1.2190</td>
<td>1.0000</td>
</tr>
<tr>
<td>c-uses coverage</td>
<td>1.1436</td>
<td>1.0000</td>
</tr>
<tr>
<td>p-uses coverage</td>
<td>1.1758</td>
<td>1.0005</td>
</tr>
</tbody>
</table>

2.b Coverage ratios on bessjy

<table>
<thead>
<tr>
<th></th>
<th>FSCS-ART vs. RT</th>
<th>ECP-FSCS-ART vs. RT</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>max</td>
<td>min</td>
</tr>
<tr>
<td>block coverage</td>
<td>1.0385</td>
<td>0.9787</td>
</tr>
<tr>
<td>decision coverage</td>
<td>1.0471</td>
<td>0.9876</td>
</tr>
<tr>
<td>c-uses coverage</td>
<td>1.0393</td>
<td>0.9786</td>
</tr>
<tr>
<td>p-uses coverage</td>
<td>1.0424</td>
<td>0.9853</td>
</tr>
</tbody>
</table>

2.c Coverage ratios on quadratic

<table>
<thead>
<tr>
<th></th>
<th>FSCS-ART vs. RT</th>
<th>ECP-FSCS-ART vs. RT</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>max</td>
<td>min</td>
</tr>
<tr>
<td>block coverage</td>
<td>1.0395</td>
<td>0.9858</td>
</tr>
<tr>
<td>decision coverage</td>
<td>1.0435</td>
<td>0.9677</td>
</tr>
<tr>
<td>c-uses coverage</td>
<td>1.0475</td>
<td>0.9836</td>
</tr>
<tr>
<td>p-uses coverage</td>
<td>1.0472</td>
<td>0.9619</td>
</tr>
</tbody>
</table>

2.d Coverage ratios on cubic

<table>
<thead>
<tr>
<th></th>
<th>FSCS-ART vs. RT</th>
<th>ECP-FSCS-ART vs. RT</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>max</td>
<td>min</td>
</tr>
<tr>
<td>block coverage</td>
<td>1.0484</td>
<td>0.9499</td>
</tr>
<tr>
<td>decision coverage</td>
<td>1.1076</td>
<td>0.9523</td>
</tr>
<tr>
<td>c-uses coverage</td>
<td>1.0398</td>
<td>0.9476</td>
</tr>
<tr>
<td>p-uses coverage</td>
<td>1.0666</td>
<td>0.9452</td>
</tr>
</tbody>
</table>

2.e Coverage ratios on tcas

<table>
<thead>
<tr>
<th></th>
<th>FSCS-ART vs. RT</th>
<th>ECP-FSCS-ART vs. RT</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>max</td>
<td>min</td>
</tr>
<tr>
<td>block coverage</td>
<td>1.0582</td>
<td>0.9218</td>
</tr>
<tr>
<td>decision coverage</td>
<td>1.0988</td>
<td>0.8722</td>
</tr>
<tr>
<td>c-uses coverage</td>
<td>1.0449</td>
<td>0.9404</td>
</tr>
<tr>
<td>p-uses coverage</td>
<td>1.0906</td>
<td>0.9224</td>
</tr>
</tbody>
</table>
been shown that ART normally has a smaller F-measure than RT, that is, ART uses fewer test cases than RT to reveal the first software failure.

In this paper, we proposed to measure the performance of ART from a different perspective. We evaluated the coverage percentages attained by ART algorithms and examined whether and to what extent ART can test the program more thoroughly than pure RT with the same number of test cases. Two particular ART algorithms were chosen in our study, namely fixed-size-candidate-set ART (FSCS-ART) and FSCS-ART with partitioning by edge and centre (ECP-FSCS-ART). We measured the control-flow and data-flow coverage of these two ART algorithms as well as pure RT through a series of experiments. The experimental results have shown that both ART algorithms generally have higher coverage percentages than RT. Moreover, ECP-FSCS-ART, which was originally proposed to enhance the failure-detection capability of FSCS-ART, also attains more thorough coverage than FSCS-ART. Such results indicate another advantage of ART over RT, that is, ART increases our confidence on the quality of the program under test by covering the program more thoroughly than pure RT with the same number of test cases. As a summary, ART is a more effective software testing approach than RT, not only because of its better failure-detection capabilities, but also due to its higher coverage percentages.

As a pilot study, this paper has investigated the coverage percentages attained by two ART algorithms on five numerical programs. It is worthwhile to continue the research by further evaluating different ART algorithms on various large-scale real-life programs, especially those with non-numeric inputs.

Acknowledgment

This research project is supported by an Australian Research Council Discovery Grant (DP0880295).

References


Telcordia Technologies. Telcordia software visualization and analysis toolsuite (xSuds).


