Using the Information:
Incorporating Positive Feedback Information into the Testing Process

Kwok Ping Chan* and Dave Towey
Department of Computer Science and Information Systems,
University of Hong Kong
{kpchan, dptowey}@csis.hku.hk

Tsong Yueh Chen, Fei-Ching Kuo and Robert Merkel
School of Information Technology,
Swinburne University of Technology
{tychen, dkuo, rmerkel}@it.swin.edu.au

Abstract

Software Testing is recognized as an essential part of the Software Development process. Random Testing (RT), the selection of input test cases at random from the input domain, is a simple and efficient method of Software Testing. RT does not however make use of previously executed test case information; in particular, information about non-failure-causing test cases is ignored. Intuitively, use of this positive feedback information can improve the failure-finding efficiency of a testing method. Adaptive Random Testing (ART) makes use of knowledge of general failure pattern types, and information of previously executed test cases, in the selection of new test cases. A failure pattern in a program’s input domain is composed of the regions of failure-causing inputs. Previous research has categorized failure patterns broadly into three types: point; strip; and block, and has identified important implications for the failure-finding efficiency of test methods, depending on the failure pattern type. In particular, it has been found that for non-point type patterns, the efficiency of RT can be improved upon by simple modification of the basic approach: by ensuring a more even and widespread distribution of test cases over the input domain, the number of test cases required to find the first failure (F-measure) can be reduced dramatically. This insight has motivated several Adaptive Random Testing methods, and produced convincing results. This paper introduces some of the research in this area and suggests areas of interest for future work.

1. Introduction

Despite the now perceived maturity of the field, Software Engineering (SE) has yet to attain the goal of producing high quality software, reliably and timely. Such is the state of affairs that, “[t]he average customer of the computing industry has been served so poorly that he expects his system to crash all the time, and we witness a massive worldwide distribution of bug-ridden software for which we should be deeply ashamed” [10].

The branch of SE associated with the bulk of the quality assurance is Software Testing, which, although widely acknowledged as a vital part of the software development process, is still a discipline very much under development. Since the testing of all possible input combinations to the Software Under Test (SUT) is possible only for the smallest of programs, the question of how best to make use of the limited testing resources is foremost in the testing process.

Testing methodologies fall broadly into two types: those that make use of information concerning the program’s structure (White Box Testing), and those that do not (Black Box Testing). A simple Black Box method is to select test cases at random from the input domain, a technique commonly referred to as Random Testing (RT). In addition to the existence of efficient test case generation algorithms [11], the ease of inferring reliability estimates and conducting statistical analyses [18] make RT an excellent candidate method, particularly in the early stages of development [12]. Moreover, given the simplicity of the method, it is a particularly appealing approach for many inexperienced testers.

One argued disadvantage of RT as a method is the waste of potentially useful, executed test case information [16]: Only in the case of Random Testing without replacement of test cases [9] is any record or use made of the executed test case data. It is the authors’ belief that, by incorporating more of the executed test case...
information, improvements over the basic RT failure-finding efficiency should be possible. In particular, positive feedback information from, for example, executed test cases which have not caused failure, should be used to guide the choice of subsequent test cases. One approach in which this intuition has been successfully applied is in the Adaptive Random Testing (ART) methods [6, 7, 9], where non-failure-causing test case information (with general knowledge of failure pattern types) is used to guide the selection of new test cases.

Antirandom Testing (AT) [14] is another Black Box test case methodology which, like ART, uses the notion of distance to space a sequence of test vectors throughout an input domain. In this approach, test vectors are represented by fixed length, binary strings. To generate a testing sequence, an initial vector (test case) is first chosen randomly, and then subsequent vectors are selected such that each maximizes the distance from all previously selected vectors. The method differs from ART methods in that it is almost completely deterministic, with randomness only applicable in the case of tie-breaking equally distant vectors. In addition, because the AT method requires the enumeration of the entire input domain, the applicability of the method to non-binary domains is somewhat restricted, and computing large Antirandom sequences is extremely time-consuming (there is, however, an approximation method [15] which attempts to reduce this overhead).

This paper introduces some of the background and current research in the area of Adaptive Random Testing (ART). In the next section, ART is explained in some detail. In Section 3, we identify some of the other areas, related to the Adaptive Random Testing methods, which are currently under investigation. Finally, in Section 4, we summarize what has been presented in this paper.

2. Adaptive Random Testing

Adaptive Random Testing (ART) is a failure pattern-based testing strategy that makes use of executed test case information. Previous research into how certain errors in software can manifest in the form of particular failure patterns (regions from which an input applied to the SUT would cause failure) has resulted in the categorization of several types of failure pattern [3]. It has been suggested that certain of these failure patterns can mean that targeted testing in the input domain may enable a faster failure-finding result than with ordinary RT approaches [9]. In studies [6, 7, 9, 13], it has been found that, under certain circumstances, by slightly modifying RT and requiring the test cases to be more widespread and evenly distributed over the input domain, the failure finding efficiency can be improved, in some cases by as much as 80% [6].

There are many possible ways of encouraging an even and widespread distribution of test cases. One way to do so was by applying the notion of “farthest” test case. By maintaining two sets of test cases (the executed and the candidate sets), each time a new test case is required, the element in the candidate set which is “farthest away” from the elements in the executed set is selected [9, 13]. Chan et al. [6, 7] use exclusion regions around executed, but non-failure-causing, test cases, restricting subsequent test cases to be drawn from outside of the exclusion regions. With the use of additional information comes additional overhead, and several adaptations of the basic ART methods have been investigated, all attempting to reduce this additional overhead [4, 8].

2.1. F-measure

In the literature, testing strategies are usually measured according to the probability of detecting at least one failure (the P-measure), or by the expected number of failures successfully detected (the E-measure). An alternative has been proposed [6, 7] the expected number of test cases required to find the first failure (the F-measure). When testing software, it is often the case that as soon as a failure is found, the testing stops; the F-measure, therefore, is not only intuitively more appealing than either the E- or P-measures, but also is more realistic from a practical viewpoint.

Following the notation of Chen et al. [9], for an input domain D, we let d denote the domain size, and m the size of the failure-causing input regions. The failure rate, \( \theta \), is then defined as the proportion of the entire input domain which is failure-causing (\( m/d \)). For Random Testing with replacement of test cases [9], the expected value for F is equal to \( 1/\theta \), or equivalently, \( d/m \). For example, when using Random Testing on an input domain with a failure rate (\( \theta \)) of 0.1%, we expect the F-measure to be 1000.
2.2. Failure Patterns

Chan et al. [3] have identified three major categories of failure patterns (input points which cause failures), and reported on how they influence the performance of some partition testing strategies. The categories are: point, which is characterised by individual or small groups of failure-causing inputs; strip, characterised by a narrow strip of failure-causing inputs; and block, characterised by the failure-causing inputs being concentrated in either one or a few regions. Examples of these failure pattern categories are given below in Figure 1. (In the figures, the shaded areas represent the failure-causing regions, and the borders represent the outer boundaries of the input domain).

![Figure 1. Types of Failure Patterns](image)

It should be noted that, although based on failure pattern types, the ART methods obviously do not require information on a particular program’s failure pattern prior to testing (were it to be known, there would be no need to test!); nor do the methods attempt to define the pattern for the Software Under Test (SUT). When exact failure pattern information is required, for example, after testing has concluded and when we wish to analyze the programs further, then alternative approaches (e.g. exhaustive execution of the input domain) are applied to the SUT. Such approaches are beyond the scope of this paper.

2.3. Motivation of Adaptive Random Testing

Chen et al. [9] suggested that the failure-finding efficiency of Random Testing methods could, for non-point type failure patterns, be greatly improved by ensuring that the test cases be far separated from each other, and more evenly distributed within the input domain. Since intuitively a program fault is likely to result in many contiguous areas being failure-causing, it is likely that point-type failure patterns will be far less common than non-point types [9]. Because of this, even though there is no available information on the failure pattern of the SUT in advance of testing, adopting a method that ensures an even distribution of test cases is appropriate.

The question of how to achieve this distribution was addressed by Chen et al. in the Distance-based Adaptive Random Testing methods (D-ART) [6, 7], and Chan et al., in the Restricted Adaptive Random Testing methods (R-ART) [6, 7]. They differ in their approaches, but both achieve excellent results in terms of improvement over RT’s failure-finding efficiency.

2.3.1. Distance-based ART (D-ART). In D-ART [9], two sets of test cases are maintained: the executed set, which contains those test cases already executed but without causing failure; and the candidate set, a set of cases selected randomly from the input domain. Each time a new test case is required, the element in the candidate set with the maximum minimum distances from all the executed set elements is selected. In the Fixed Size Candidate Set Implementation of D-ART (FSCS), a completely new candidate set is generated each time an element is selected. In simulations, this method has been seen to outperform RT by an average of about 40%, in terms of the calculated F-measure for each method!

2.3.2. Restricted ART (R-ART). With R-ART [6, 7], the input domain from which test cases are generated is restricted to only those regions not close to previously executed test cases. This is implemented by creating an exclusion zone around each previously executed, but non-failure-causing input. The exclusion zone is the same size for each test case, but the area around each decreases with subsequent executions. Two versions of R-ART exist: Ordinary R-ART (OR-ART) [7]; and Normalized R-ART (NR-ART) [6]. The NR-ART version incorporates a scaling/homogenizing feature that reduces the effect that the input domain shape has on the efficacy of the algorithm. In simulations, the R-ART method has been seen to outperform RT by an average of about 40%, in terms of the calculated F-measure for each method!

2.4. Empirical Studies of Adaptive Random Testing

To investigate the efficiency of the ART methods, they were applied to several error-seeded programs. In each study, the number of test cases required to find the first failure (F-measure) for the different methods (including ordinary Random Testing (RT)) was calculated and, the percentage improvement over Random Testing determined.

2.4.1. Error-seeded Programs. The R-ART methods have been tested against seven error-seeded programs. These are published programs [1, 17], all involving numerical calculations, which were written in C (converted to C++) and which varied in length from 30 to 200 statements.
Using Mutation Analysis [2], errors in the form of simple mutants were seeded into the different programs. Four types of mutant operators were used to create the faulty programs: arithmetic operator replacement (AOR); relational operator replacement (ROR); scalar variable replacement (SVR) and constant replacement (CR). These mutant operators were chosen since they generate the most commonly occurring errors in numerical programs [3]. For each program, after seeding in the errors, the range of each input variable was then set such that the overall failure rate would not be too large [3]. Table 1 summarizes the details of the error-seeded programs.

2.4.2. R-ART Applied to Seeded Programs. The results of the R-ART methods when applied to the error-seeded programs are given in Figure 2. The figure shows the percentage improvement over the Random Testing (RT) F-measure for both the OR-ART and NR-ART versions of Restricted ART (R-ART) [6, 7].

With the exception of the sncndn program, which has a point-type failure pattern [6, 7] and is therefore unlikely to show improvement with more evenly spread test cases, the results for the ART methods show a significant improvement over RT, with up to 80% improvement in two cases for NR-ART.

Table 1. Program name, dimension (D), input domain, seeded error types, and total number of errors for each of the error-seeded programs

<table>
<thead>
<tr>
<th>Program Name</th>
<th>D</th>
<th>Input Domain</th>
<th>Error Type</th>
<th>Total Errors</th>
</tr>
</thead>
<tbody>
<tr>
<td>bessj</td>
<td>2</td>
<td>(2.0, -1000.0) (300.0, 15000.0)</td>
<td>2 1 1 1</td>
<td>4</td>
</tr>
<tr>
<td>bessj0</td>
<td>1</td>
<td>(-300000.0) (300000.0)</td>
<td>2 1 1 1</td>
<td>5</td>
</tr>
<tr>
<td>cel</td>
<td>4</td>
<td>(0.001, 0.001, 0.001, 0.001) (1.0, 300.0, 10000.0, 1000.0)</td>
<td>1 1 1</td>
<td>3</td>
</tr>
<tr>
<td>erfcc</td>
<td>1</td>
<td>(-300000.0) (300000.0)</td>
<td>1 1 1 1</td>
<td>4</td>
</tr>
<tr>
<td>gammq</td>
<td>2</td>
<td>(0.0, 0.0) (1700.0, 40.0)</td>
<td>3 1</td>
<td>4</td>
</tr>
<tr>
<td>plgndr</td>
<td>3</td>
<td>(10.0, 0.0, 0.0) (500.0, 11.0, 1.0)</td>
<td>1 2 2</td>
<td>5</td>
</tr>
<tr>
<td>sncndn</td>
<td>2</td>
<td>(-5000.0, -5000.0) (5000.0, 5000.0)</td>
<td>4 1</td>
<td>5</td>
</tr>
</tbody>
</table>

Figure 2. Improvement in F-measures for the Ordinary Restricted ART (OR-ART) and Normalized Restricted ART (NR-ART) methods compared with the Random Testing (RT) F-measure. All figures refer to the improvement over the calculated RT.
3. What Next?

Adaptive Random Testing represents a way in which information about executed test cases, in particular non-failure-causing test cases, can be used to improve basic testing methods. Much evidence has been produced supporting the benefits of testing methods incorporating this kind of information [6, 7, 9], and it is believed that there are potentially many other ways that this information can be used. In this section we discuss briefly some of the research areas which we believe may yield additional benefits for testers. We also address some of the overhead issues associated with the presented methods and outline some research into alleviating these costs.

3.1. Widespread Distribution of Test Cases

Both D-ART and R-ART are based on the goal of achieving a widespread and even distribution of test cases over the input domain. Alternative methods of achieving this goal are also currently under investigation: the use of fuzzy features and approaches seems particularly hopeful [5], but other alternatives are also sought.

3.2. Failure Patterns

Chen et al. [3] identified 3 types of failure patterns. Subsequent investigation has observed how certain testing strategies find failure regions faster than other strategies according to the failure pattern [6] of the tested program. Further work in this area is called for. Of particular interest will be: other categories of pattern; frequency of occurrence in programs; and correlation of the pattern with the causing error.

The strategies adopted in D-ART and R-ART were targeting a non-point type failure pattern. As other patterns are enumerated, alternative goals may become more important in strategies attempting to find failure regions. This in turn will lead to fresh investigation into methods for achieving the new goals.

3.3. Overheads

Both D-ART and R-ART, by incorporating the additional information, are also increasing the computational overheads of the testing process. Alternative approaches, maintaining the failure-finding efficiency while reducing the computations required, are currently under study. Chan et al. [4] have been investigating an aging factor in R-ART, according to which only a number of the most recent test cases (instead of all) would be considered in the algorithm. Preliminary results suggest that this method may indeed have similar results to the original R-ART method, but with significantly less overhead.

Chen et al. [8] propose a mirroring method, M-ART, whereby the original algorithm (D-ART or R-ART) is performed only in a subdomain of the original input domain, and successive test cases are mapped, using a mirror function, to the other subdomains. Preliminary investigations and simulations show significant reduction in overhead with comparable failure-finding efficiency.

3.4. Non-numerical Programs

So far, both D-ART and R-ART have only been applied to numerical programs, specifically those for which Random Testing (RT) is an option. The possibility of extending the algorithm to be applicable to other domains is also currently under investigation.

4. Summary

Software Testing is an essential part of Software Development. A very simple method of testing is to select test cases at random from the input domain, a method referred to as Random Testing (RT). A criticism of RT, and several other methods, is the waste of the potentially useful information available to the tester. When this information, e.g., data on non-failure-causing input points, is incorporated into the testing, and used to guide subsequent test case selection, significant improvements in the testing process are possible.

Adaptive Random Testing (ART) chooses test cases based on the executed test case information, and knowledge of failure patterns types (regions of failure-causing inputs in a program’s input domain). There are different ways of using executed test case information, and ART, has several variations, all with the same goal but differing in implementation. Other possible formulations are under study.

With the use of additional information in the test case selection, comes additional computation and overhead. Various methods aimed at reducing this overhead are also under study.

Although research into using this positive feedback information about non-failure-causing test cases is still at an early stage, given the very encouraging results already obtained, it promises to continue being an interesting area and a powerful tool for the tester!
Acknowledgement

We should like to gratefully acknowledge the support given to this project by the Research Grants Council of the Hong Kong Special Administrative Region, China (Project No: HKU 7007/99E), and the Discovery Project Grant of the Australian Research Council (DP0345147).

References