Adaptive Random Testing: The ART of test case diversity

Article *in* Journal of Systems and Software · January 2010

DOI: 10.1016/j.jss.2009.02.022 · Source: DBLP

4 authors, including:

Fei-Ching Kuo
Swinburne University of Technology
86 PUBLICATIONS **1,406** CITATIONS

SEE PROFILE

Robert G. Merkel
Monash University (Australia)
38 PUBLICATIONS **763** CITATIONS

SEE PROFILE

T.H. Tse
The University of Hong Kong
200 PUBLICATIONS **3,761** CITATIONS

SEE PROFILE

HKU CS Tech Report TR-2009-07

To appear in *Journal of Systems and Software*

Adaptive Random Testing: the ART of Test Case Diversity☆

Tsong Yueh Chen[a], Fei-Ching Kuo[a], Robert G. Merkel[a], T.H. Tse[b]

[a]*Faculty of Information and Communication Technologies, Swinburne University of Technology, John St., Hawthorn 3122, Australia.*
[b]*Department of Computer Science, The University of Hong Kong, Pokfulam, Hong Kong.*

Abstract

Random testing is not only a useful testing technique in itself, but also plays a core role in many other testing methods. Hence, any significant improvement to random testing has an impact throughout the software testing community. Recently, *Adaptive Random Testing* (*ART*) was proposed as an effective alternative to random testing. This paper presents a synthesis of the most important research results related to ART. In the course of our research and through further reflection, we have realised how the techniques and concepts of ART can be applied in a much broader context, which we present here. We believe such ideas can be applied in a variety of areas of software testing, and even beyond software testing. Amongst these ideas, we particularly note the fundamental role of diversity in test case selection strategies. We hope this paper serves to provoke further discussions and investigations of these ideas.

Key words:
software testing, random testing, adaptive random testing, adaptive random sequence, failure-based testing, failure pattern

1. Introduction

Despite decades of effort to develop alternative technologies, software testing remains the primary way to verify the quality of software systems. However, it remains a labour-intensive, slow and imperfect process. It is, therefore, important to consider how testing can be performed more effectively and at a lower cost through the use of systematic, automated methods.

Attempts to automate the generation of test data through various forms of random selection date from the early 1960's (Renfer, 1962), and have been a regular feature of the research literature and industrial practice. Random testing is simple in concept, often easy to implement, has been demonstrated to effectively detect failures, is good at exercising systems in unexpected ways (which may not occur to a human tester), and may be the only practical choice when the source code and the specifications are unavailable or incomplete. It has been used extensively as a testing method in itself; furthermore, it forms a core part of many other testing methods. On the other hand, it is often argued that such random testing is inefficient, as there is no attempt to make use of any available information to guide testing. This work now encompasses a family of methods in which random selection can play a greater or lesser part.

A growing body of research has examined the concept of *Adaptive Random Testing* (*ART*), which is an attempt to improve the failure-detection effectiveness of random testing. ART is based on various empirical observations showing that many program faults result in failures in contiguous areas of the input domain, known as *failure patterns*. ART systematically guides, or filters, randomly generated candidates, to take advantage of the likely presence of such patterns. In this paper, we attempt to provide a synthesis of the growing body of piecewise research in the area of ART. We provide new insights and interpretations by explicitly drawing out novel links among individual research results. We report how ART can be adapted

for testing a wide variety of types of software, and how it has inspired theoretical analysis that has revealed fundamental connections between software failure behaviour and maximum testing effectiveness. We believe that this work, viewed as a whole, is significant both to researchers in software testing and to practitioners.

Section 2 introduces the concept of failure patterns, which provided the key inspiration for ART, the fundamentals of which are described in Section 3. Under certain conditions, ART is close to an optimally effective test case selection strategy, as we outline in Section 4. Section 5 describes how ART can be applied to a wide range of software through the development of appropriate "distance measures". In Section 6, our attention turns from summarising work already conducted to exploring possibilities for broadened research by applying the lessons learned from ART. We conclude with a brief summary in Section 7.

1.1. A note on "randomness"

Typically, pseudorandom sequences, generated by standard deterministic methods, are used instead of truly non-deterministic random numbers when "random testing" is being conducted in research and in real life practice. While pseudorandom sequences are, by definition, not truly random, all published articles on random testing to date treat such pseudorandom sequences as equivalent to random. For simplicity, therefore, we shall refer to testing according to such pseudorandom sequences as "random" in this paper.

When conducting random testing, testers may choose an appropriate sampling distribution to meet their requirements. When trying to accurately estimate the delivered reliability of software, for instance, testers may choose to sample according to a profile that reflects the expected usage profile of the software — known as an *operational profile*. On the other hand, analysis of random testing as a testing strategy has normally assumed a uniform sampling profile. Throughout this paper, unless otherwise specified, we also assume a uniform sampling profile.

2. Failure Patterns

Essentially, the testing process can be viewed as taking samples from the set of all possible inputs to the software under test (known as the *input domain*), executing the samples one by one, and determining whether the outputs from each sample match the software specification. If the outputs do not match the specification, a software *failure* is revealed. The presence of a software failure implies the existence of a *fault* — an actual code defect in the software concerned. (Obviously, many software failures can be related to the same fault.) A tester seeks to select test data with a view to maximising the number of distinct faults detected. To help the tester in this task, it is natural to consider how faults may cause different parts of the input domain to produce erroneous outputs when executed — in other words, reveal failures.

A pioneering work in this area was that of White and Cohen (1980), who analysed certain types of program fault in numerical programs. They observed that when the contents of predicates (decision-making points in the source code) were erroneous, an incorrect computation path would be taken (referred to as *domain errors*). This would, therefore, often result in contiguous regions of the input domain that reveal failures. White and Cohen then proposed a systematic technique for detecting such errors.

More empirical studies came to similar conclusions about the tendency for software faults to result in contiguous "failure regions" within the input domain. Ammann and Knight (1988) analysed a number of sample numerical programs to determine the distribution of failures caused by various faults. In their small sample, they found that the faults resulted in "locally continuous" failure regions. A more comprehensive study was conducted by Bishop (1993), who examined program faults in control functions for nuclear reactors. He found that virtually all the faults were "blob" faults — that is, each fault revealed failures in a contiguous region of the input domain.

Chan et al. (1996) also noted that certain common types of fault in numerical software would also lead to typical distributions of failure-causing inputs throughout the input domain, which they termed *failure patterns*. They categorised three such patterns: (i) the *block pattern*, where failures form a locally compact, contiguous region of the input domain; (ii) the *strip pattern*, similar to the patterns resulting from White and Cohen's domain errors, in which a "strip", contiguous but elongated along one or more dimensions, would reveal failures; and (iii) the *point pattern*, where failures would spread in a non-contiguous manner throughout the input domain. They argued that strip and block failure patterns were much more common than point patterns.

All of these quite different studies lead to a more general conclusion: that, in numerical programs, many program faults lead to contiguous failure regions of the program input domain.

$T = \{\}$ /* T is the set of previously executed test cases */
randomly generate an input t
test the program using t as a test case
add t to T
while (stopping criteria not reached)
    $D = 0$
    randomly generate next k candidates $c_1, c_2, \ldots, c_k$
    for each candidate c_i
        calculate the minimum distance d_i from T
        if $d_i > D$
            $D = d_i$
            $t = c_i$
    add t to T
    test the program using t as a test case
end while

Figure 1: FSCS-ART algorithm.

3. Adaptive Random Testing

If contiguous failure regions are indeed common, it would suggest that one way to improve the failure-detection effectiveness of random testing is to somehow taking advantage of this phenomenon.

One corollary of the existence of contiguous failure regions is that "non-failure regions", that is, regions of the input domain where the software produces outputs according to specification, will also be contiguous. Therefore, given a set of previously executed test cases that have not revealed any failures, new test cases located away from these old ones are more likely to reveal failures — in other words, test cases should be more evenly spread throughout the input domain. Based on this intuition, Adaptive Random Testing (ART) was developed to improve the failure-detection effectiveness of random testing.

The first ART method proposed, the Fixed Size Candidate Set ART algorithm (FSCS-ART) (Chen et al., 2004), is described in Figure 1. Essentially, to choose a new test case, k candidates are randomly generated. For each candidate c_i, the *closest* previously executed test is located, and the distance d_i is determined. The candidate with the largest d_i is selected, and the other candidates are discarded. The process is repeated until the desired stopping criterion, be it the exhaustion of testing resources or the detection of enough failures, is reached.

Figure 2 shows FSCS-ART in operation, on a program with a two-dimensional input space such that $k = 3$. In Figure 2(a), we show four previously executed test cases, t_1 to t_4. We wish to select an additional test case, so three candidates, c_1 to c_3 are randomly generated as shown. To choose among the candidates, we must calculate d_i for each. Figure 2(b) depicts this process for candidate c_1, and Figure 2(c) shows the nearest t_i for each candidate. The dashed lines in Figure 2(c) indicate d_i for the respective candidates. We choose the candidate with the *largest* d_i, which is c_2 in this example. Thus, we discard candidates c_1 and c_3, treat c_2 as test case t_5, and execute it. We repeat the process until the stopping criterion is reached.

To assess the effectiveness of the FSCS-ART method, Chen et al. compared the failure detection effectiveness of FSCS-ART to random testing — that is, testing by uniform random sampling with replacement — on a sample of 12 error-seeded numerical programs. The original, unmodified programs were used as a testing oracle to check the correctness of the outputs. The statistic used to compare the methods was the average number of tests required to detect the first failure, which is commonly known as the *F-measure*. In most cases, the F-measure of FSCS-ART was 30–50% lower than that of random testing. Results of simulations using a variety of failure patterns, with different failure rates and geometries, are consistent with the experimental results.

While such improvements are significant, it is reasonable to speculate that there might be other, more efficient ways to take advantage of contiguous failure regions which would result in an even smaller F-measure. A number of different methods, using different intuitions to achieve "even spread", have been investigated in the literature. One such is Restricted Random Testing (RRT), which is based on the notion of exclusion to achieve the even spreading fundamental to ART (Chan et al., 2006). It involves the creation of "exclusion zones" around test cases that have been executed. A randomly generated input will be used as the next test case if it lies outside all exclusion regions; otherwise it will be discarded and the process will be repeated. The effectiveness of RRT is very similar to that of FSCS-ART. ART by Partitioning (Chen et al., 2004) uses a rather different intuition — in essence, that partitioning the input domain, and allocating test cases evenly to partitions, will achieve even spread. Other attempts to take advantage of failure region contiguity, but using various other intuitions to achieve the "even spreading" of test cases, include Quasi-Random Testing (Chen and Merkel, 2007), and Lattice-Based ART (Mayer, 2005).

Interestingly, all of these methods have similar ranges of failure-detection performance, with the maximum

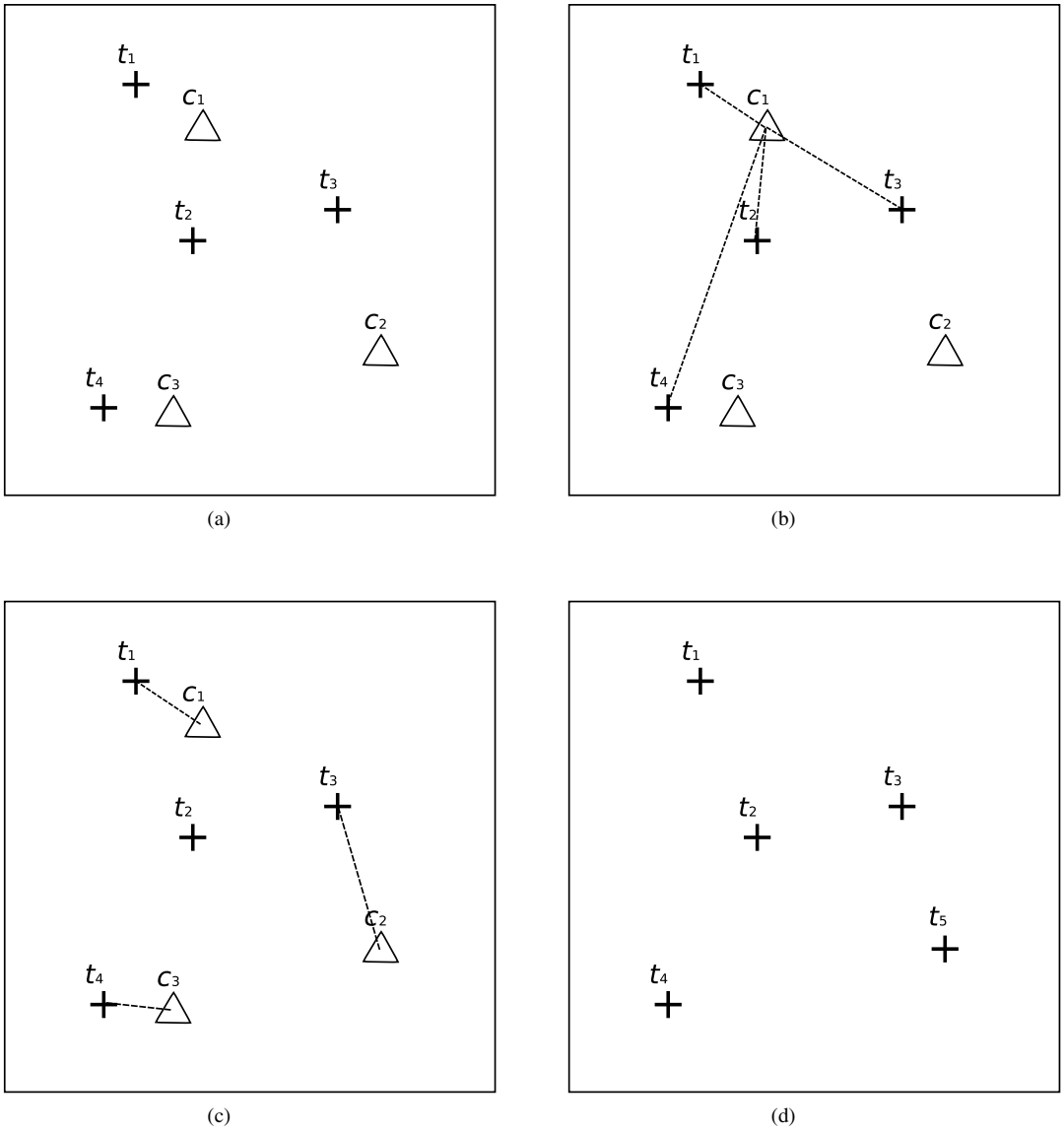

Figure 2: FSCS-ART in operation. Previously-executed test cases are denoted by crosses, and randomly generated candidates are denoted by triangles. To select a new test case, (a) multiple candidates are randomly generated; (b) the nearest previously-executed test case to each candidate is determined; (c) these nearest distances are compared among all candidates; and (d) the candidate with the *longest* such distance is selected.

improvement over random testing being around 50%. However, the different methods perform best under various circumstances. For instance, some methods offer lower selection overheads, or work well when the dimensionality of the input domain is large.

Antirandom testing (Malaiya, 1995) is another testing method that uses a related concept of "distance" to distribute test cases. However, there are several major differences between it and ART. Antirandom testing is almost exclusively deterministic; the only non-determinism comes in the choice of the first test case in the set. Furthermore, the method requires the number of test cases to be chosen in advance, unlike the flexibility of incremental generation offered by ART.

4. Theoretical Limits

If so many different approaches to taking advantage of failure contiguity achieve similar results, an interesting question arises — is the failure to make further improvements a lack of imagination by researchers in identifying better methods? Are existing ART methods too similar to one another — might an entirely different approach achieve better results? Or are existing solutions close to optimally effective already? In the computer science world, such questions are traditionally answered by theoretical complexity analyses of problems. We set out to apply the same approach to this problem — how much can we improve on random testing by using failure contiguity information? We have proved (Chen and Merkel, 2008) that there is indeed a fundamental limit to how much failure contiguity information, when used on its own, can improve failure-detection effectiveness.

Our approach to doing so is simple in principle. We first consider a case where the tester has more information about the failure pattern than available in reality — in essence, the tester knows that there is one single, contiguous *failure region* of the input domain. The tester knows the size, shape and orientation of this single failure region — everything about it except *where* it is located in the input domain. In fact, the tester does not have *any* information about the location of the failure region in the input domain. Laplace's Principle of Indifference (Keynes, 2006) states that, if a decision maker knows the possible states of the world, and truly has no information about the plausibility of each possible state, they should act as if each state were equally likely. In this context, as the tester has no information about the location of the failure region, they should assume that it is equally likely to be located in any possible location within the input domain. The tester has strictly more information about failure contiguity than that assumed by various ART algorithms, and absolutely no information about the failure region location, which is also assumed in ART.

Given these assumptions, we then devise an optimal strategy for selecting test cases, and show definitively that it will have an F-measure *lower than or equal to any other strategy* (recalling that the F-measure is an *average*). In essence, we create a "grid" of test cases at regularly spaced locations throughout the input domain, and execute the resulting tests in an arbitrary order. On particular occasions, such a strategy might "get lucky" and reveal a failure on the first test case. Over many trials, however, the F-measure of such a strategy will be at least half that of the F-measure of random testing with replacement, given the same failure rate.

That is, **no** *strategy using failure pattern information, other than information about its location, can reduce F-measures by more than 50% compared to random testing.*

This result still holds even if there are multiple, contiguous failure regions. The proof is complex, but based on the same principles as the single failure region case. Interested readers may refer to (Chen and Merkel, 2008).

The implications of our result are quite clear. ART, which uses strictly less information, still often achieves effectiveness improvements which are quite close to the maximum theoretically possible. Therefore, any further improvements in the testing effectiveness of ART must come from taking account of additional information about the program's failure location. Alternatively, rather than attempting to improve failure-detection effectiveness, researchers can develop ART methods that have lower overheads in evenly spreading the test cases, in order to improve the overall cost-effectiveness. Furthermore, the closeness of ART's performance to the theoretical bound indicates that the bound is indeed a tight one.

5. ART beyond Numeric Programs

Initial studies of ART showed that it can improve the failure-detection effectiveness of random testing substantially, and this improvement is indeed close to the theoretical maximum possible (in the absence of further information on failure location). Nevertheless, these initial studies were limited to software with numeric inputs. Much, perhaps most, software of practical interest does not have such simple input parameters. It is therefore of considerable importance to study how to apply ART to a broader class of programs.

As an illustration of the broader application of ART, we again consider FSCS-ART. To apply FSCS-ART in a given situation, two issues must first be resolved: a method to sample randomly from the input domain of the software under test, and some way to compare any two members of the input domain and determine the "distance" between them. The first issue is not unique to ART: by definition, pure random testing also requires the ability to sample randomly from the input domain! In practice, random generation of test cases can be a challenging problem. However, the generation of random test cases of sufficient quality to reveal significant software faults has been thoroughly studied and demonstrated in numerous application domains such as SQL servers (Slutz, 1998, Bati et al., 2007) and Java Virtual Machines (Yoshikawa et al., 2003), among many others. Hence, we shall not address this issue further in this paper.

By contrast, the second issue — a "distance" measure — is unique to ART. The algorithm will execute given any trivial distance measure — such as simply returning a distance of zero, regardless of locations of the members of the input domain in question. In such cases, however, the algorithm degenerates into a more costly version of pure random testing. Therefore, in designing an appropriate distance measure, we need to consider why contiguous failure patterns occur in numeric programs, and how this concept might be generalised for a wider range of software.

Essentially, the "distance" measure needs to provide an estimation of the likelihood of two inputs having common failure behaviour — the smaller the distance, the more likely they will trigger the same failure behaviour. In fact, the "distance" measure is really a *difference* measure. Studies revealing contiguous failure patterns in numeric input domains show that adjacent test cases (as reflected by the Cartesian distance measure) were likely to result in similar computations. In turn, it is our intuition that the similarity of computation is a good predictor of the similarity of failure behaviour. To apply ART effectively in a non-numeric context, therefore, alternative methods to measure the similarity of computation resulting from the executions of two test cases are required.

We have proposed a difference measure (Kuo, 2006, Merkel, 2005) that can be applied to a broad range of software input types, based on the concepts of categories and choices proposed by Ostrand and Balcer (1988) for the *category-partition method*. The category-partition method is a specification-based testing method. The tester must first identify the parameters and environment conditions determining the behaviour of the software under test, known as *categories*. For each category, *choices* are defined as mutually exclusive sets of values which are expected to trigger similar computation.

Our work makes use of the concepts of categories and choices as the basis of a difference measure for ART, allowing ART to be applied to a broader range of software. Intuitively, the more categories in which two inputs have different choices, the more different will be the computation they trigger. Therefore, a count of categories with differing choices can be used as a difference measure.

As an illustration, consider a simple object recognition system, which can distinguish shapes, sizes and colours. Suppose that the colour of objects can only be light-red, red, deep-red, light-blue, blue, deep-blue, light-green, green and deep-green, and objects are spheres, cubes or pyramids in shape. The size is in the range $(0, 10]$ in m^3. The system behaviour depends only on the object shape, the "base colour" — red, blue or green, and whether the object is larger than $1m^3$. In this case, we can define three categories: Colour, Shape and Size; three choices for the Colour category: [red], [blue] and [green]; three choices for the Shape category: [sphere], [cube] and [pyramid]; and two choices for the Size category: [large] and [small]. Some choices contain more than one possible value. For example, the [red] choice has light-red, red and deep-red as its possible values and [large] has any size more than $1m^3$.

Consider two program inputs T_1 and T_2, where T_1 is a light-red sphere of size 3.2 m^3, and T_2 is a deep-blue sphere of size 2.7 m^3. T_1 has the choices [red], [sphere] and [large] while T_2 has the choices [blue], [sphere] and [large]. In this case, therefore, there is only one category — colour — in which T_1 and T_2 differ, so the difference between the two inputs is 1 using our measure.

We have used this distance measure as the basis for the development of new ART algorithms for non-numeric software. We have used these new algorithms in several case studies including the Unix command-line utility "grep", and other programs from the UNL Software-artifact Infrastructure Repository (Rothermel et al.) Details can be found in Barus et al. (in preparation)

While we have demonstrated that it is possible to construct meaningful difference measures for a broad range of input types, this is not the only feasible approach. Recently, Ciupa et al. (2008) proposed an alternative type of difference measure in the context of object-oriented software. They provide a method for computing *object distance* between two arbitrary

objects. They first define some distance measures for elementary types, such as numbers, Booleans, strings and references. Next, they describe how to measure distances between composite objects, made up of three elements — the *type distance*, based on the difference between the two object types, the *field distance* — the distance between the matching fields, and the *recursive distance* — the distance between matching reference attributes. This method has the advantage that it completely specifies how to calculate the difference measure, supporting the complete automation of the method. However, further empirical research will need to be conducted to determine its effectiveness.

6. Further Implications

In most previously published work, ART has mainly been envisaged as an enhanced replacement of pure random testing; research has demonstrated a variety of ways in which this can be done efficiently, and shown that it can be applied to a broad range of software. We believe that ART is now sufficiently mature to be regarded as an effective alternative to random testing, and by summarising existing results, we hope to draw broader attention to it and encourage its application and further enhancement.

However, in the course of our research and through a process of reflections, we have realised how ART techniques and concepts can be applied in a much broader, more general context. In this section, we will highlight these ideas, which we believe can be applied in a variety of aspects of software testing, and possibly in other contexts beyond testing.

6.1. Adaptive random sequences

The use of random sequences is very common in many contexts in both industrial testing practice and the research literature. Random sequences are very straightforward to generate, and remove any human bias from the ordering. Therefore, random testing, or random ordering of the elements of a test suite, is commonly used as a baseline for comparison with more complex strategies. Typically, such strategies are based on selecting tests to achieve some criterion which the researcher believes correlates well with testing effectiveness; a comparison with random testing — or random ordering of a test suite — is then used to support the intuition.

In the context of test case selection, ART has been designed as a more effective replacement for random testing. Given that ART retains most of the virtues of random testing, and offers near-optimum effectiveness, it is therefore appealing to investigate the use of ART as a baseline method instead of random testing. Hence, given that random ordering is also commonly used as a baseline, ART can also be used for ordering purposes. That is, instead of using ART to *generate* its own sequence of test cases, ART can also be used to *order* a given test suite, aiming at increasing the chance to detect failures earlier. We define a so-ordered sequence as an adaptive random sequence, or AR sequence.

One obvious application for AR sequences is for regression testing. In regression testing, a large test suite may be accumulated over time, even to the extent that not all of them can necessarily be run each time a change is made. Hence, various techniques have been developed to prioritise the elements of a test suite, based on a number of different criteria. We believe that AR sequences may be a simple, effective and relatively low-overhead alternative. Furthermore, there are many other testing techniques (such as path testing techniques) that can generate a larger set of test cases than can be run with available resources; AR sequences may be very useful in these circumstances.

It may even be that AR sequences have uses beyond testing. Quasi-random sequences (Chen and Merkel, 2007), which have been proposed as an alternative to ART for testing purposes, are used in a wide variety of contexts. AR sequences may have similarly wide applications. Quasi-random sequence generation is intimately tied to the properties of the binary representation of floating-point numbers (Bratley et al., 1992); AR sequences are in fact more easily applicable to a wider range of data types. Furthermore, standard quasi-random sequence generation algorithms only generate very few distinct sequences; AR sequencing can trivially be used to generate large numbers of distinct sequences.

6.2. Failure-based testing

ART is based on the notion that software failures manifest themselves in contiguous regions in the input domain. Therefore, we view ART as an example of a *failure-based testing* technique. Some earlier techniques, such as White and Cohen's domain testing (1980), implicitly take advantage of failure pattern information, notwithstanding its original conception as a fault-based technique. However, ART is, as far as we are aware, the first testing technique explicitly designed as a failure-based testing method.

We define a failure-based testing method as one that selects test cases based on the knowledge about various aspects of failure patterns, such as shapes, sizes,

locations and numbers. ART is a relatively simple failure-based testing method, as it takes advantage of only one form of knowledge about failure patterns — the fact that they are often contiguous within the program input domain. It is not difficult to conceive of other failure-based testing methods that take advantage of other information, either as an alternative to or in addition to contiguity information. As a simple example, it is widely known that failures are likely to manifest themselves on or near boundaries between subdomains. A testing strategy that selects test cases near or on subdomain boundaries could therefore also be viewed as a failure-based testing technique.

A contrast here should be made between failure-based testing and the well-established concept of *fault-based testing*. Fault-based testing describes a class of testing strategies designed to demonstrate that a certain type of fault is either absent or present in a program. The domain testing strategy of White and Cohen (1980), for instance, is a classic example of a fault-based testing technique. The domain testing strategy guarantees to reveal faults in predicates in numeric software. Failure-based testing does not look directly at faults themselves, rather their manifestations as failures within the program input domain. It is therefore necessarily a looser concept than fault-based testing.

However, for both types of testing strategies, empirical research plays an important role. In the case of fault-based testing, empirical research can provide the necessary information to prioritise the search for particular fault classes; if a class of faults very commonly occurs in practice, searching for them is obviously a higher priority. Similarly, effective failure-based testing must be based on empirical research about typical failure patterns, including their geometry and distribution. We believe that such research can serve as the basis for the development of new failure-based testing techniques, and the refinement of existing ones. One complicating factor is, of course, that failure patterns for non-numeric inputs are defined on the basis of particular difference measures as discussed in Section 5; research on such programs will need to take this into account.

From another perspective, testing based on failure patterns can be viewed as a type of specialised search problem. In this view, feedback from the tests as they are executed guides the continuing search for failure-causing input. ART is a simple and successful realisation of this concept. Many testing methods regard tests that do not reveal a failure as, essentially, wasted, but ours is not the only work to disagree with this view. For instance, metamorphic testing (Chen et al., 1998) uses previously executed "original" test cases to construct "follow-up" test cases. The relationship between the outputs for the original and follow-up test cases is then checked to verify program correctness. This approach is designed specifically to alleviate the oracle problem. Pacheco et al. (2007) take advantage of non-failure-causing test cases as "feedback", by using past test cases that do not reveal failure as building blocks to construct more complex ones. *Search-based testing* also seeks to use feedback from past test cases to guide future test case selection, but generally much more selectively. Most such work to date has taken into account additional information from test execution, such as execution paths, to help guide the search, and often only considers the most recent few test cases rather than the entirety. Without such guidance, conventional search algorithms will not be able to tell where to "go next", so it will be very challenging to adapt these techniques for failure-based testing. Research into the geometry and distribution of failure patterns will help in the design of appropriate search algorithms. These algorithms could take into account the results of *many* previous test cases, thus making best use of the limited information available from each test case.

6.3. A theory of software testing

Our theoretical analysis showing that ART performs close to the theoretical maximum is significant in itself. However, the approach used to show this is also worthy of further consideration. While different types of theoretical analyses have been conducted for different testing strategies, we believe that our approach is novel and can serve as a model to build a deeper understanding of the relationship between failure information and software testing.

There have been a number of theoretical analyses of various coverage criteria. These analyses have shown that achieving one type of coverage may imply another coverage — a trivial example is that branch coverage implies statement coverage. Such analyses are useful, but they do not directly correlate to failure-detection capabilities. By contrast, in fault-based testing, fault *subsumption* relationships for certain fault classes have been developed (Kapoor and Bowen, 2007). As such, a hierarchy of certain fault types has been described, and the subsumption relationships among fault-based testing strategies have been explored. There have also been a number of papers that evaluate individual testing techniques for failure-detection effectiveness, or compare two testing techniques. These tend to be

quite specific in their applicability, such as proving a sufficient condition for one testing technique to be more effective than another (Chen and Yu, 1996, Morasca and Serra-Capizzano, 2004).

Our approach was quite different from all of the above. We explicitly develop a model about the tester's prior knowledge about failures, and identify the best performance that can be achieved by *any* testing strategy using only this information. This not only evaluates the performance of known strategies, but can also be applied to testing strategies that have not yet been invented. In this way, it can help to identify where methodological research should best be applied to improve the state of the art, similarly to how time and space complexity analysis is useful to algorithm researchers in computer science.

It might well be possible to use this general approach to identify other relationships between information available and testing effectiveness. Ultimately, a complexity hierarchy that relates various types of information about the software under test, to a class of testing strategies, may be possible. For such a class, the best practical strategies developed to date, and theoretical effectiveness bounds, can be identified. Our work represents a first step towards such a hierarchy. We also note it is unlikely that any such hierarchy would be as elegant or comprehensive as the complexity class hierarchy of problems in theoretical computer science. Nevertheless, we believe that the development of such a hierarchy will have a significant impact on the theory of software testing, and will identify where methodological research can best be directed.

6.4. The role of test case diversity

The key intuition that led us to develop ART was the concept of "even spreading" throughout the input domain. We have come to realise that "even spreading" can be better described as a form of *diversity*. For the numeric case, at least, neighbouring inputs tend to result in similar computations. An even spread of test cases throughout the input domain, therefore, gives rise to a diversity of computations.

While the importance of diversity is hardly a new or surprising insight, ART achieves a very simple form — perhaps the simplest possible form — of test case diversity. There have been a variety of different notions of test case diversity intrinsic in some testing methods over the years; for instance, the different types of control coverage and dataflow coverage criteria yield test sets with different notions of diversity. Ultimately, in testing, the tester seeks diversity in failure behaviour, so that test cases reveal as many different ways in which the program can fail with the given testing resources. As failure behaviour information cannot ever be completely available before the test cases are executed, testers must find various other models of diversity that strongly correlate with failure behaviour.

In the study of partition testing, the proportional sampling (PS) strategy (Chen and Yu, 1996) stipulates that the number of randomly selected test cases from each partition should be proportional to the corresponding partition size. This strategy provides a sufficient condition for partition testing to have its probability of detecting at least one failure not smaller than that for random testing with replacement.

Chen et al. (Chen et al., 2001) observed that "a comparison of ART with PS strategy reveals an interesting similarity; the PS strategy can be regarded as a form of ART. Such a similarity between the PS strategy and ART appears striking ... the two strategies were initially proposed for very different reasons. The PS strategy was motivated by the need of providing a universally safe strategy [which is guaranteed to outperform random testing], whereas ART attempts to improve random testing in those situations where the failure-causing inputs tend to cluster ... no distribution of the failure-causing inputs was assumed when deriving the PS strategy." This interesting similarity can now be interpreted as being due to their common "diversity over the input domain". Hence, we believe that a new way to classify test case selection strategies may be based on various forms of diversity.

ART achieves diversity not only in the context of the entire test suite, but also within the subset of test cases executed at any one time. When an AR sequence is used to order test suites that already exhibit diversity according to some specific criterion, the current subset of executed tests, at any stage of testing, exhibits additional *local* diversity. Such local diversity will improve the chances of detecting failures earlier.

7. Conclusion

Based on empirical observations that contiguous failure regions are common, adaptive random testing combines random candidate selection with a filtering process to encourage an even spread of test cases throughout the input domain. Experimental studies have shown that ART can detect failures using up to 50% fewer test cases than random testing. In fact, ART methods achieve close to the theoretical maximum test case effectiveness by any possible testing method using the same information. Early work on ART concentrated

mainly on numeric input domains; however, recent research has shown that it can also be applied to a broad range of software. As such, we believe that it represents an effective, efficient alternative to random testing in many applications.

On the other hand, research on ART may have broader implications, and we have discussed a number of them in this paper. The AR sequence is a promising, general method of incremental ordering. The success of ART illustrates the potential of the approach of failure-based testing, and the impact and importance that diversity has on the effectiveness of test suites. Our theoretical work, motivated by ART, paves the way for a more rigorous and scientific analysis of the relationships between the information available to the software tester and the effectiveness of families of testing strategies — *including those not yet developed*. We believe such an analytic approach will provide a significant contribution to the foundations of software testing.

Acknowledgement

This work was supported in part by a Discovery Grant of the Australian Research Council (project no. ARC DP 0880295) and the General Research Fund of the Research Grants Council of Hong Kong (project no. 717308).

We would also like to thank our colleagues who have worked with us on adaptive random testing over the years: A. Barus, K.P. Chan, G. Eddy, D.H. Huang, H. Leung, H. Liu, I.K. Mak, G. Rothermel, K.Y. Sim, C.A. Sun, D.P. Towey, P.K. Wong, W.E. Wong, and Z.Q. Zhou.

References

Ammann, P.E., Knight, J.C., 1988. Data diversity: an approach to software fault tolerance. IEEE Transactions on Computers 37 (4), 418–425.

Barus, A.C., Chen, T.Y., Kuo, F.-C., Merkel, R.G., Rothermel, G., in preparation. Adaptive random testing of programs with arbitrary input types: a methodology and an empirical study.

Bati, H., Giakoumakis, L., Herbert, S., Surna, A., 2007. A genetic approach for random testing of database systems. In: Proceedings of the 33rd International Conference on Very Large Data Bases (VLDB 2007). VLDB Endowment, pp. 1243–1251.

Bishop, P.G., 1993. The variation of software survival time for different operational input profiles (or why you can wait a long time for a big bug to fail). In: Digest of Papers, the 23rd International Symposium on Fault-Tolerant Computing (FTCS-23). IEEE Computer Society Press, Los Alamitos, CA, pp. 98–107.

Bratley, P., Fox, B.L., Niederreiter, H., 1992. Implementation and tests of low-discrepancy sequences. ACM Transactions on Modeling and Computer Simulation 2 (3), 195–213.

Chan, F.T., Chen, T.Y., Mak, I.K., Yu, Y.T., 1996. Proportional sampling strategy: guidelines for software testing practitioners. Information and Software Technology 38 (12), 775–782.

Chan, K.P., Chen, T.Y., Towey, D.P., 2006. Restricted random testing: adaptive random testing by exclusion. International Journal of Software Engineering and Knowledge Engineering 16 (4), 553–584.

Chen, T.Y., Cheung, S.C., Yiu, S.M., 1998. Metamorphic testing: a new approach for generating next test cases. Technical Report HKUST-CS98-01, Department of Computer Science, Hong Kong University of Science and Technology, Hong Kong.

Chen, T.Y., Eddy, G., Merkel, R.G., Wong, P.K., 2004. Adaptive random testing through dynamic partitioning. In: Proceedings of the 4th International Conference on Quality Software (QSIC 2004). IEEE Computer Society Press, Los Alamitos, CA, pp. 79–86.

Chen, T.Y., Leung, H., Mak, I.K., 2004, Adaptive random testing, In: Proceedings of the Ninth Asian Computing Science Conference (ASIAN'04), Lecture Notes in Computer Science, Vol 3321, 320–329.

Chen, T.Y., Merkel, R.G., 2007. Quasi-random testing. IEEE Transactions on Reliability 56 (3), 562–568.

Chen, T.Y., Merkel, R.G., 2008. An upper bound on software testing effectiveness. ACM Transactions on Software Engineering and Methodology 17 (3), Article No. 16.

Chen, T.Y., Tse, T.H., Yu, Y.T., 2001. Proportional sampling strategy: a compendium and some insights. Journal of Systems and Software 58 (1), 65–81.

Chen, T.Y., Yu, Y.T., 1996. On the expected number of failures detected by subdomain testing and random testing. IEEE Transactions on Software Engineering 22 (2), 109–119.

Ciupa, I., Leitner, A., Oriol, M., Meyer, B., 2008. ARTOO: adaptive random testing for object-oriented software. In: Proceedings of the 30th International Conference on Software Engineering (ICSE 2008). ACM Press, New York, NY, pp. 71–80.

Kapoor, K., Bowen, J.P., 2007. Test conditions for fault classes in Boolean specifications. ACM Transactions on Software Engineering and Methodology 16 (3), 1–12.

Keynes, J.M., 2006. A Treatise on Probability. Cosimo, New York, NY.

Kuo, F.-C., 2006. On Adaptive Random Testing. PhD Thesis, Swinburne University of Technology, Melbourne, Australia.

Malaiya, Y.K., 1995. Antirandom testing: getting the most out of black-box testing. In: Proceedings of the 6th International Symposium on Software Reliability Engineering (ISSRE '95). IEEE Computer Society Press,

Los Alamitos, CA, pp. 86–95.

Mayer, J., 2005. Lattice-based adaptive random testing. In: Proceedings of the 20th IEEE/ACM International Conference on Automated Software Engineering (ASE 2005). ACM Press, New York, NY, pp. 333–336.

Merkel, R.G., 2005. Analysis and Enhancements of Adaptive Random Testing. PhD Thesis, Swinburne University of Technology, Melbourne, Australia.

Morasca, S., Serra-Capizzano, S., 2004. On the analytical comparison of testing techniques. In: Proceedings of the 2004 ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA 2004), ACM SIGSOFT Software Engineering Notes 29 (4), 154–164.

Ostrand, T.J., Balcer, M.J., 1988. The category-partition method for specifying and generating functional tests. Communications of the ACM 31 (6), 676–686.

Pacheco, C., Lahiri, S.K., Ernst, M.D., Ball, T., 2007. Feedback-directed random test generation. In: Proceedings of the 29th International Conference on Software Engineering (ICSE 2007). IEEE Computer Society Press, Los Alamitos, CA, pp. 75–84.

Renfer, G.F., 1962, Automatic program testing. In: Proceedings of 3rd Conference of the Computing and Data Processing Society of Canada. University of Toronto Press, Toronto, Canada.

Rothermel, G., Elbaum, S., Kinneer, A. Do, H., Software-artifact Infrastructure Repository. Available at `http://sir.unl.edu`.

Slutz, D.R., 1998. Massive stochastic Testing of SQL. In: Proceedings of the 24th International Conference on Very Large Data Bases (VLDB '98). Morgan Kaufmann, San Francisco, CA, pp. 618–622.

White, L.J., Cohen, E.I., 1980. A domain strategy for computer program testing. IEEE Transactions on Software Engineering SE-6 (3), 247–257.

Yoshikawa, T., Shimura, K., Ozawa, T., 2003. Random program generator for Java JIT compiler test system. In: Proceedings of the 3rd International Conference on Quality Software (QSIC 2003). IEEE Computer Society Press, Los Alamitos, CA, pp. 20–24.

Tsong Yueh Chen received his BSc, and MPhil degrees from the University of Hong Kong; MSc degree and DIC from Imperial College of Science and Technology; and PhD degree from the University of Melbourne. He is currently a Professor of Software Engineering in the Faculty of Information and Communication Technologies, Swinburne University of Technology, Australia. His research interests include software testing, debugging, software maintenance, and software design.

Fei-Ching Kuo is currently a Lecturer at Swinburne University of Technology, Australia. She received her PhD degree in Software Engineering, and BSc (Honours) in Computer Science, both from the Swinburne University of Technology. Her research interests include software testing, debugging and project management. She has been an IEEE member for many years, a PC member of several international conferences and workshops, including SAC and QSIC amongst others, and also acted as a reviewer for several international journals, including the Journal of Systems and Software and Journal of Software.

Robert Merkel received his BSc (Hons) from the University of Melbourne and his PhD from Swinburne University of Technology. He is currently a Lecturer in Software Engineering at Swinburne in the Faculty of Information and Communication Technologies. His current main research interests include software testing, and program verification. Prior to his PhD study, he worked in industry on an open source software project.

T.H. Tse is a Professor in Computer Science at The University of Hong Kong. His research interest is in software testing and analysis. He is an editorial board member of Software Testing, Verification and Reliability and Journal of Systems and Software, the steering committee chair of QSIC and a standing committee member of COMPSAC. He is a fellow of the British Computer Society, a fellow of the Institute for the Management of Information Systems, a fellow of the Institute of Mathematics and its Applications and a fellow of the Hong Kong Institution of Engineers. He was decorated with an MBE by Queen Elizabeth II of the United Kingdom.