Adaptive Random Testing: An Illusion of Effectiveness

Andrea Arcuri
Simula Research Laboratory
P.O. Box 134, 1325 Lysaker, Norway
arcuri@simula.no

Lionel Briand
Simula Research Laboratory and University of Oslo
P.O. Box 134, 1325 Lysaker, Norway
briand@simula.no

ABSTRACT

Adaptive Random Testing (ART) has been proposed as an enhancement to random testing, based on assumptions on how failing test cases are distributed in the input domain. The main assumption is that failing test cases are usually grouped into contiguous regions. Many papers have been published in which ART has been described as an effective alternative to random testing when using the average number of test case executions needed to find a failure (F-measure). But all the work in the literature is based either on simulations or case studies with unreasonably high failure rates. In this paper, we report on the largest empirical analysis of ART in the literature, in which 3727 mutated programs and nearly ten **tr**illion test cases were used. Results show that ART is highly inefficient even on trivial problems when accounting for distance calculations among test cases, to an extent that probably prevents its practical use in most situations. For example, on the infamous Triangle Classification program, random testing finds failures in few milliseconds whereas ART execution time is prohibitive. Even when assuming a small, fixed size test set and looking at the probability of failure (P-measure), ART only fares slightly better than random testing, which is not sufficient to make it applicable in realistic conditions. We provide precise explanations of this phenomenon based on rigorous empirical analyses. For the simpler case of single-dimension input domains, we also perform formal analyses to support our claim that ART is of little use in most situations, unless drastic enhancements are developed. Such analyses help us explain some of the empirical results and identify the components of ART that need to be improved to make it a viable option in practice.

Categories and Subject Descriptors

D.2.5 [**Software Engineering**]: Testing and Debugging;
I.2.8 [**Artificial Intelligence**]: Problem Solving, Control Methods, and Search

General Terms

Algorithms, Experimentation, Reliability, Theory

Keywords

Faulty region, random testing, F-measure, P-measure, shape, diversity, similarity, distance.

1. INTRODUCTION

Random testing does not exploit any information about the software under test (SUT). Therefore, historically it was considered a naive testing strategy that is less effective than partition testing. Myers in his 1979 book [34] states "probably the poorest [test case design] methodology is random input testing ...". But this statement was not agreed upon by everyone. Thayer *et al.* [41] for example were favorable to random testing, recommending it as a necessary final step of the testing activity.

In a seminal paper, Duran and Ntafos [17] carried out a series of experiments in which they showed that random testing could be more effective than the commonly used partition testing. That was a counterintuitive result that opened the doors to a large body of literature on the properties and benefits of random testing. Random testing has been shown to be a very useful tool in the hands of software testers and it is not a minor technique as it was originally thought [17, 5].

Usually, in random testing the test cases are sampled at random from the input domain D of the SUT with uniform probability. In other words, each test case is sampled with probability $p = 1/|D|$. The motivation is that, if we do not know anything about the SUT, then each test case is as likely as the others to be able to detect failures. However, if the distribution of failing test cases follows some specific patterns, then this information could be exploited to give higher probability to sampling test cases that are more likely to detect faults.

One property that has been noticed in numerical applications is that often failing test cases tend to cluster in *contiguous regions* of the input domain D [42, 1, 19, 38], referred to as *faulty regions*. For example, in a function that takes as input two integers x and y, if there is a fault inside a conditional statement such as `if(x>=0 && y>=0 && x>=y)`, then we could have a contiguous region of failing test cases in the positive sub-area $x \geq y$.

Based on the above assumption, a modified version of random testing has been proposed: Adaptive Random Testing (ART) [13, 12]. The fundamental idea behind ART is to reward *diversity* among sampled test cases. If a test case does not detect any failure, then in presence of contiguous faulty regions it would be better to sample test cases that are *far* from it. The *distance* among test cases in a input domain D depends on the type of SUT. In case of numerical inputs, the Euclidean distance can be used [13, 12].

Although the contiguous region hypothesis has been evaluated in the literature [42, 1, 19, 38], the role that the *shapes* of these faulty regions plays on ART's effectiveness has not received enough at-

tention. In fact, most work on ART has been mainly evaluated with simulations where no actual SUT was employed (e.g., [11, 27, 40]). In these simulations, a very limited number of types of shapes were considered, with practically no empirical evaluation of how often these shapes appear in actual SUTs. When empirical analyses are carried out (e.g., [8, 9]), there is no attempt to correlate the shape of the faulty regions in these SUTs with the performance of ART in a *systematic* way. In fact, to gain precise knowledge of these shapes, executing test cases on the entire input domain D is required, and that would be particularly expensive. Knowing the properties of these shapes in real SUTs would be essential to develop new variants of ART that exploit such information.

To obtain more information on how shapes appear in actual SUT, we carried out a large empirical analysis on 11 programs. For each program, a series of *mutants* were generated to introduce faults in these programs in a systematic way [2]. Faults generated through mutation allows us to generate a large number of faults, in an unbiased and varied manner. We generated 3727 mutants and selected the 780 of them with lower detection probabilities to carry out our empirical analysis of faulty region shapes, following a selection process further described later in the paper. For *each* of the 3727 mutated programs, for practical reasons we limited their input domain to carry out an exhaustive analysis of 16.7 million test cases. To the best of our knowledge, this is the largest empirical analysis of ART to date.

Because the number of failing test cases in the entire case study is very high, we designed novel metrics to automatically characterize the properties of the failing region shapes. We then *tried* to carry out an empirical analysis on each of the 780 mutants to compare ART with random testing. The novel metrics were used to assess the relationship between the faulty region shapes of these mutants and the performance of ART in terms of the average number of executed test cases required to obtain a failure (F-measure). However, as soon as we tried to empirically compare ART with random testing, we found ART to be an *extremely slow* technique. On trivial problems that are solved by random testing in few milliseconds, ART requires several minutes or even hours instead.

The very bad performance of ART can be explained by simply looking at its computational and memory complexity, without actually even needing to run any experiment. The reason why this simple fact has not been addressed in the literature has a controversial explanation: All the analyses were using simulations based on unrealistic assumptions. In the few cases in which empirical analyses were carried out, only a limited number of programs with manually injected faults were used. This led to very biased case studies. Last but not least, these analyses were only considering the F-measure and ignoring the very heavy overhead of calculating distance values in ART. Even when considering the less frequently used P-measure, the probability of triggering at least one failure when running a test set of fixed size, though ART expectedly fares better than random testing, the difference is not sufficient to have a practical impact when considering realistic failure rates. A small number of randomly selected test cases, relatively to the size of the input space, are unlikely to be close to each other, or at least close enough to make a difference.

To provide a better insight on the behavior of ART, in this paper we also theoretically analyze the role of faulty region shapes in numerical SUTs with one integer as input (i.e., $n = 1$). Based on the results of the theoretical analysis, we have developed a deterministic approach that is better than ART for fixed number of sampled test cases. Our motivation is to use the performance of this algorithm as an upper bound for ART. Based on a theoretical analysis, we show that ART cannot perform nearly as well as a deterministic algorithm in the one-dimension domain, explain why that is the case and, more importantly, conjecture that such a phenomenon is likely to also occur in larger dimensions.

Z={}
add random test case to Z and execute it
repeat until stopping criterion is satisfied
  sample set W of random test cases
  for each w of these $|W|$ test cases
    $w.minD = min(dist(w,z \in Z))$
  execute and add to Z the w with maximum $minD$

Figure 1: Pseudo-code of ART.

The paper is organized as follows. Section 2 gives a brief introduction to ART. Section 3 presents new metrics to enable wide-scale analysis of the shapes of faulty regions, for example based on mutation analysis. Empirical analyses using these metrics on high dimensional input domains can be found in Section 4, along with comparisons of random testing and ART using the F-measure and the P-measure. In an attempt to provide theoretical insights, a formal analysis of the one-dimension input domain follows in Section 5 along with a discussion of how results might generalize to higher dimensions. The generalization of these empirical and theoretical results is discussed in Section 6. Section 7 describes relevant work in the literature in which ART has been used in other testing domains than numerical applications. Finally, Section 8 concludes the paper.

2. ADAPTIVE RANDOM TESTING

ART was first proposed by Chen *et al.* [13] in 2004. ART works like random testing, with the difference that at each step a set W of test cases is randomly sampled (a typical value is $|W| = 10$, see [13]). Only the most diverse test case in W is executed and added to the current set Z of test cases. Diversity is calculated with all the test cases in Z executed so far, and the used distance measure is problem dependent. The basic algorithm of ART is depicted in Figure 1. If ART stops after evaluating ρ test cases, then the number of distance calculations between test cases is $\sum_{i=1}^{\rho-1} |W|i = |W|\rho(\rho-1)/2 = \Theta(\rho^2)$.

Over the years, several improvements on the basic ART algorithm have been proposed. For example, there has been work to reduce the number of distance calculations [11, 32, 26, 6], to improve the sampling distribution of the test cases [8, 9], to better handle higher dimensional input domains [27], to combine ART with operational profiles [10], to combine ART with evolutionary algorithms [40], to understand the role of faulty region shapes [38], to identify the conditions for which ART should work better than random testing [7], and to apply ART to integration testing [39]. These works mainly concentrate on numerical applications [12], but ART has recently started to be used in other testing domains, such as unit testing in Object-Oriented software [15, 29], test case prioritization [25, 44], model-based test case selection [24] and system testing of embedded systems [4]. Moreover, theoretical analyses to understand the limitations of ART have been performed [14].

In the literature, to compare the effectiveness of ART with random testing, the F-measure is what is usually employed [12]. The F-measure is simply the average number of test cases sampled by a testing technique before obtaining a test case that reveals a failure. Given a failure rate θ, because random testing follows a geometric distribution [18], then the F-measure of random testing is $1/\theta$. Re-

ported empirical work on ART shows that its F-measure can get as close as to half the F-measure of random testing, i.e., on average ART is no more than twice as fast as random testing. This has also been formally proven under some conditions [14].

Notice that, besides ART, there are also other types of extensions/enhancements of random testing, as for example Randoop [35] and DART [20]. For reasons of space and because these techniques are not based on maximizing diversity of the input data, which is our focus here, we do not analyze them in this paper. A general framework to analyze variants of random testing is an important research objective that we leave for future work. The results presented in this paper are a further step toward such a direction.

3. FAULTY REGION SHAPE METRICS

There is empirical evidence that fault revealing test cases tend to cluster in contiguous regions [42, 1, 19, 38], in which some patterns of shapes have been found. However, we are aware of only one work in the literature that tries to analyze the faulty region shapes in a systematic way [38]. Three programs were analyzed using image analysis and clustering techniques.

In this paper, we propose two novel metrics to evaluate the faulty region shapes that are relatively quick to compute. These metrics are meant to be used to analyze the performance of ART on different types of shapes. Because these analyses involve an exhaustive evaluation of the input domain, particular care needs to be taken to achieve as low a computational complexity as possible. Since a metric is necessarily a compromise as it cannot fully describe a particular shape without involving an impractical amount of computation, we discuss the limitations of our proposed metrics.

The first metric we introduce is $M1$, and it is defined as:

$$M1 = \frac{f}{\prod_{i=1}^{n} g(i) + |g(i) - f^{1/n}|} ,$$

where f is the total number of failing test cases, and $g(i)$ is the number of distinct values in the input dimension i that appear at least once in a failing test case. For example, if in a two dimension domain (i.e., $n = 2$) we have $(x = 1,y = 3)$ and $(x = 4,y = 3)$ as the only failing test cases, then $g(1) = 2$ and $g(2) = 1$. To calculate $g(i)$ we just need to sort the failing test cases based on their values in their ith dimension, and then we can simply linearly scan these values to compute $g(i)$.

In several simulations (see [12]), it has been noticed that usually ART shows optimal performance when there is only one faulty region, and that faulty region extends with identical length in all the n dimensions (for example, if $n = 2$ then the faulty region shape is a square, and a cube if $n = 3$). Let us call R this type of regular shape. The metric $M1$ yields 1 if the faulty region extends with same length in all dimensions. Otherwise the further the faulty region from the R shape, the smaller $M1$, which remains strictly positive. If R, it is easy to see that $\prod_{i=1}^{n} g(i) = f$. Because R has to extend with identical length in all directions, then all the $g(i)$ should be the same, so $g(i) = f^{1/n}$. Therefore, $M1(R) = f/(\prod g(i) + 0) = 1$. For other types of shapes, Figure 2 shows some examples for $n = 2$ and $f = 4$.

The metric $M1$ does not take into account the number of faulty regions in the input domain, which could be more than one. This might affect the performance of ART. We hence introduce a metric $M2$, with values between 0 and 1, that takes into account the number of disconnected faulty regions $\mathcal{FR}$ and that tells us with certainty that there are more than one faulty region when its value is less than one. For each dimension i, we calculate $h(i)$ as follows: sort the f failing test cases based on the values of the ith dimension, and then calculate the number of times a value v for which there is a failing test case is followed by a value k for which a failing test case exists but $k \neq v + 1$. For example, if $n = 2$ and the f failing test cases are $(x = 1,y = 1)$, $(x = 1,y = 3)$ and $(x = 1,y = 4)$, then $h(1) = 0$ and $h(2) = 1$. It easily follows that:

$$\mathcal{FR} \geq max(h(i)) + 1 .$$

For the previous example, we have $\mathcal{FR} \geq 2$, which is correct, as there are only two distinct faulty regions in this case. We use $M2 = 1/\prod_{i=1}^{n}(h(i) + 1)$. If there is only one faulty region, then we are certain that $M2 = 1$. If there are more than one faulty region, $M2$ may however wrongly suggest that there is one region and can therefore be considered a higher bound. However, $M2$ is still useful in large scale empirical studies to identify the situations where we can be certain that there is more than one faulty region.

Notice that it is possible to design metrics that calculate the exact number of faulty regions, but with a much higher time complexity. The metric $M2$ just needs to sort the values in the different domains and is therefore in the order $\Theta(f \log f)$. Metrics that require the comparison of each failure revealing test case with all the other ones would be in the order $\Theta(f^2)$. When exhaustive analyses are involved, moving from complexity $\Theta(f \log f)$ to $\Theta(f^2)$ could be infeasible for high values of f.

Finally, we can combine $M1$ and $M2$ in $M = M1 \times M2$. This metric yields 1 if there is only one single faulty region ($M2 = 1$) of type R ($M1 = 1$), and otherwise a smaller positive number based on how many distinct faulty regions there are and on how different their shapes are from R.

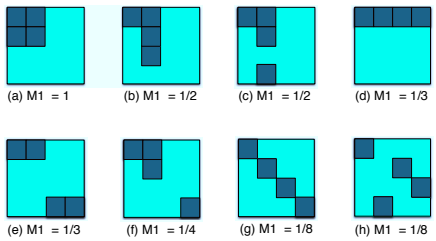

Figure 2: Examples of $M1$ measure on 2-dimensional domain with four failing test cases.

4. CASE STUDY

In this paper we consider 11 numerical programs for the empirical analysis. These programs include basic mathematical routines [16, 36] and most of the programs used previously in the ART literature [38]. All the programs have been written or translated in Java.

Though these programs are small, our goal is to compare our results with those already published. Our study uses programs of sizes comparable to existing studies and the expected impact of using larger programs will be subsequently discussed. Table 1 summarizes some of the programs' properties.

4.1 Case Study Design

For each program, we used the tool muJava [30] to generate mutant versions. Mutation is an effective technique to inject a large number of faults in a systematic and unbiased way [2]. All the method level operators of muJava were used and 3727 mutants were generated.

As shown in Table 1, some of the programs take as input an array. In these cases, we used arrays of length 4. Each integer

Table 1: Properties of the case study. LOC stands for lines of code.

Name	LOC	If/Loops	Input
Triangle	26	6/0	int,int,int
Triangle2	41	10/0	int,int,int
Median	20	1/3	int[]
Remainder	48	5/4	int,int
Bessj	131	9/2	int,double
Variance	22	0/2	int[]
BubbleSort	14	1/2	int[]
Encoder	65	2/1	byte[]
Expint	86	7/3	int,double
Fisher	71	6/2	int,int,double
Gammq	89	9/3	double,double

Table 2: Properties of the generated mutants.

Name	Total	Equivalent	Timeout	Easy	Appropriate
Triangle	191	34	0	95	62
Triangle2	333	47	0	118	168
Median	89	6	11	72	0
Remainder	382	290	37	54	1
Bessj	1007	111	15	560	321
Variance	71	10	2	59	0
BubbleSort	66	3	10	53	0
Encoder	283	123	0	160	0
Expint	432	46	14	188	184
Fisher	615	0	16	599	0
Gammq	258	0	1	213	44

input is constrained in the range $[0,2^{24/n}-1]$, where n is the number of dimensions. For each of the 3727 mutants, we exhaustively generated and ran $2^{24} \approx 16.7$ million test cases, leading up to roughly 62 **b**illion test case executions on a computer cluster. Because some mutations could lead to programs that never halt, we put a 10 minute time limit on the experiments of each mutated programs. In other words, the execution of the 16.7 million test cases on a program would be stopped after 10 minutes if still running. Fortunately, such cases were only a few in our empirical analysis.

If a mutant is not killed by any test case, then the mutation might have just created a semantically equivalent program. However, given the large number of such mutants, and though we run a large number of test cases, we cannot be completely sure that these mutants are actually equivalent. We nevertheless use the expression "equivalent" to indicate these cases.

To investigate the types of faulty region shapes, we only used a subset of the 3727 mutants. We excluded the equivalent mutants and the ones that led to computation timeouts. Furthermore, we excluded the mutants that were too *easy* to kill. The threshold was arbitrarily set to $\theta > 0.01$. With such failure rates, since random testing follows a geometric distribution [18], one would need on average less than 100 random test cases to find the first failure. In our case study, generating and running 100 test cases only takes on average less than a millisecond. This left us with 780 "appropriate" mutants, roughly a fifth of the original set. Table 2 summarizes the types of these mutants for each of the 11 programs.

For each appropriate mutant, we calculated the $M1$ and $M2$ measures on their failing test cases to obtain information about the failing region shapes. Results for $M1$ are summarized in Table 3, ordered by failure rate ranges, those failure rates being computed based on exhaustive testing.

4.2 Faulty Region Shapes and F-measure Performance

We found out that 683 of the 780 mutants had $M2 = 1$, of which only 16 mutants had $M1 = 1$, which are listed in the last row of Table 3. Note that only one test case triggers a failure in each of these 16 mutants, hence explaining that $M1 = 1$. Excluding these 16 cases, we have not found a single case in which a faulty region extends with the same length in all dimensions. $M1$ values mostly lie within a $[10^{-4},10^{-1}]$ range (see Table 3). The $M2$ results means that only 97 mutants (about 12 percent) have for sure more than one faulty region, thus confirming previous studies that have shown that faulty test cases tend to cluster in contiguous regions of the input domain.

We performed an empirical investigation to compare the performance of ART, in terms of the F-measure, with random testing on each of the 780 appropriate mutants. But we were not able to do it, since ART was *extremely* slow already on the first problem (i.e., the Triangle Classification). One of the mutants generated for Triangle had a failure rate $\theta = 1.51 \cdot 10^{-5}$. We ran ART and random testing on it, 30 times each using different random seeds. Executions were stopped when they found the first failure. The original program was used as the oracle. We collected statistics about the number of sampled test cases and the time required to run these algorithms. Table 4 summarizes these results. Differences in execution times are staggering, showing orders of magnitude differences between random testing and ART, which are due to the overhead of distance computations. We also carried out Mann-Whitney U tests at a 0.05 significant level to test the statistical significance of the differences in number of test cases and execution time (for all the statistical tests in this paper we used the statistical tool R [37]). There is no statistical difference for the number of sampled test cases (p-value = 0.65), whereas, as expected, there is strong statistical difference in the execution time required by the algorithms (p-value = $2.95 \cdot 10^{-11}$). Notice that the highest computational time for random testing (14.3 milliseconds) in Table 4 is lower than the lowest value for ART (403 milliseconds). The machine used for these experiments was a MacBook Pro, 2.66 GHz, 4 giga-byte memory and 3 mega-byte L2 cache.

We cannot exclude that there are faults or inefficient code in our ART implementation, but based on our investigations there are precise explanations for the very bad performance of ART. Let us assume that we have a SUT with failure rate θ. The expected number of test cases sampled by random testing before triggering a failure is $1/\theta$. In the best conditions, according to the literature we would have ART sample, on average, half the test cases of random testing, therefore $1/(2\theta)$. The number of distance calculations would be $|W| \cdot \frac{1}{2\theta} \cdot (\frac{1}{2\theta} - 1)/2 = \Theta(\frac{1}{\theta^2})$. With a computational cost of $\Theta(\frac{1}{\theta^2})$, ART has much higher asymptotic complexity than random testing with $\Theta(\frac{1}{\theta})$. Empirical analyses are required in order to study, under realistic conditions, the failure rate threshold above which ART becomes better than random testing in terms of test execution time. However, given the overhead of distance computations, such a threshold is very likely to be unrealistically high.

Although techniques have been proposed to reduce the number of distance calculations [11, 26], their asymptotic complexity is still $\Theta(\frac{1}{\theta^2})$. The number of distance calculation is $\Theta(\frac{1}{\theta})$ in lattice-based ART [32, 6], but that technique suffers of heavy memory consumption overhead. Notice the number of distance calculations is independent from the chosen distance function (e.g., Euclidean

Table 3: $M1$ measure on the case study.

Failure Rate θ	Mutants	$M1$				
		min	median	mean	max	standard deviation
$[10^{-3},10^{-2}]$	201	$3.01 \cdot 10^{-4}$	$6.27 \cdot 10^{-4}$	$1.00 \cdot 10^{-2}$	$1.52 \cdot 10^{-1}$	$3.13 \cdot 10^{-2}$
$[10^{-4},10^{-3}]$	153	$6.20 \cdot 10^{-5}$	$7.87 \cdot 10^{-3}$	$1.94 \cdot 10^{-2}$	$2.36 \cdot 10^{-1}$	$4.23 \cdot 10^{-2}$
$[10^{-5},10^{-4}]$	22	$2.01 \cdot 10^{-6}$	$7.12 \cdot 10^{-6}$	$7.70 \cdot 10^{-3}$	$3.76 \cdot 10^{-2}$	$1.25 \cdot 10^{-2}$
$[10^{-6},10^{-5}]$	15	$8.22 \cdot 10^{-6}$	$8.22 \cdot 10^{-6}$	$8.28 \cdot 10^{-6}$	$8.36 \cdot 10^{-6}$	$6.78 \cdot 10^{-8}$
$[10^{-7},10^{-6}]$	373	$1.18 \cdot 10^{-1}$	$3.99 \cdot 10^{-1}$	$3.51 \cdot 10^{-1}$	$5.83 \cdot 10^{-1}$	$1.02 \cdot 10^{-1}$
$[10^{-8},10^{-7}]$	16	1.00	1.00	1.00	1.00	0.00

Table 4: Comparison of ART with random testing on one mutant for the Triangle program.

Data	Algorithm	min	median	mean	max	standard deviation
Test Cases	Random Testing	2,046	32,074	72,237	444,839	97,869
	ART	1,075	46,545	56,382	224,815	57,475
Time	Random Testing	0 ms	4.5 ms	10.3 ms	58 ms	14.3 ms
	ART	403 ms	19.4 minutes	47.7 minutes	5.9 hours	82.3 minutes

distance).

In the case of Triangle Classification discussed in Table 4, we had $\theta = 1.51 \cdot 10^{-5}$. This means an expected $66{,}225$ test cases for random testing, whereas in the optimal conditions (which depend on the region shape) ART would require $33{,}112$ test cases on average. But the number of distance calculations would be 5,482,022,719, which is a very high number compared to the number of sampled test cases. On this toy problem, random testing needs on average 10 milliseconds to find the first failure, whereas ART requires 47 minutes! It may be argued that in situations where test case execution takes a long time, ART might be a viable option. Using the above example further, even in optimal conditions where ART leads to half the number of test cases obtained with random testing, test execution time should be on average 1.65×10^5 larger than distance computation time for ART to pay off. For example, assuming the distance calculation time is one millisecond, that would entail roughly three minutes of test execution. We will return to this discussion in Section 6.

The number of distance calculations is not the only problem. In fact, we need to keep in mind the memory consumption of keeping a data structure for Z (set of test cases in Figure 1). In some cases, for Triangle we had $\theta = 5.96 \cdot 10^{-8}$, which on average would lead to Z containing up to roughly 8 million test cases. Even if we ignore the memory overhead of the chosen data structure for Z (e.g., a linked list), we would still need 4×3 bytes per test case (unless we use some compression algorithm). This leads to a memory consumption that can be in the order of 96 mega-bytes, which is unlikely to be stored on a first level cache. For each sampling of test case, there would be a large number of page faults in memory that would significantly increase the execution time of ART. In contrast, random testing does not require any support data structure.

In Table 3 we can see that more than half of the mutants show failure rates lower than 10^{-5}, which means than on those mutants ART is not applicable because far too expensive compared to random testing. Furthermore, the 11 problems used in the case study (Table 1), though representative of the reported ART studies, can be considered *small*, and not really comparable with real-world industrial software. In real-world software we would hence expect lower failure rates, which would make ART even less applicable.

One important question is why the problems raised by our study were not raised in previous works. Table 5 summarizes relevant properties of a (non-exhaustive) selection of widely referenced ART papers regarding numerical applications, in which ART was applied and compared against random testing. Given that the failure rate can be estimated with the inverse of the F-measure for random testing, the main flaws in existing studies can be summarized as follows:

- The simulations were based on unrealistic assumptions, as for example very high failure rates (see θ values in Table 5).
- In the cases in which empirical analyses were carried out, these included only a small number of programs. The faults in these programs were manually introduced, in an unsystematic way. This led to biased case studies in which only high failure rates were present.
- All the comparisons between ART and random testing were based on the F-measure, i.e., the average number of sampled test cases. This does not account for the *extremely high cost* of ART that is due to the distance calculations and its memory consumption.
- Empirical analyses on numerical applications using the F-measure requires the presence of an automated oracle, because it is not reasonable to ask a software tester to manually verify the output of (hundreds of) thousands of test cases. But no work in Table 5 uses a case study in which an automated oracle is available.

One of the few reported systematic studies is reference [33], in which different variants of ART were compared. Faults were automatically injected with muJava. But no information of failure rates can be inferred (the F-measure of random testing is not reported). Furthermore, the analysis of actual execution time was carried out only up to 2000 test cases, stopping before the first failure was found.

4.3 ART Performance based on the P-measure

Most of work in the literature considers the F-measure (expected number of test cases to find the first failure) instead of using the P-measure (probability of finding at least one failure given a set of test cases) [12]. But in practical contexts, the use of the F-measure

Table 5: A selection of relevant papers in the literature of ART for numerical applications.

Year	Venue	Simulations	Empirical Analyses	Lowest θ	Automated Oracle	Reference
2004	Conf.	No	Yes	$3.26 \cdot 10^{-4}$	No	[13]
2004	IST	Yes	No	$5.00 \cdot 10^{-4}$	No	[11]
2005	Conf.	Yes	No	$5.00 \cdot 10^{-4}$	No	[32]
2006	Conf.	Yes	Yes	Unspecified	No	[33]
2007	Work.	No	Yes	$1.00 \cdot 10^{-4}$	No	[38]
2007	IJSEKE	Yes	Yes	$5.00 \cdot 10^{-5}$	No	[7]
2008	SQJ	Yes	No	$5.00 \cdot 10^{-4}$	No	[27]
2008	JSS	Yes	Yes	$5.00 \cdot 10^{-5}$	No	[8]
2009	JSS	Yes	Yes	$1.00 \cdot 10^{-5}$	No	[9]
2009	TR	Yes	No	$5.00 \cdot 10^{-5}$	No	[10]
2009	TR	Yes	No	$1.00 \cdot 10^{-3}$	No	[40]
2009	Conf.	Yes	No	$2.50 \cdot 10^{-3}$	No	[26]
2009	Conf.	Yes	No	$1.00 \cdot 10^{-3}$	No	[6]
2010	Work.	Yes	No	$1.00 \cdot 10^{-3}$	No	[39]
2010	JSS	No	No	-	No	[12]

can be misleading: running an algorithm until it finds the first failure is possible only if an automated oracle is available. Often, automated oracles are not available for numerical applications, apart from checking whether the application does crash (e.g., a segmentation fault). Though the issue is usually not discussed, in most existing empirical analyses (Table 5) no realistic automated oracle is used. Correctness of faulty programs is checked against a correct version of the program. Though we have done so as well in the experiments in this paper, our objective was only to demonstrate the overhead of ART, not to assess its applicability.

If no automated oracle is available, it makes sense to evaluate ART on sets of test cases that would be manually evaluated and assuming a fixed size, driven in practice by test budgets. In other words, given a budget of $|K|$ test cases, what is the P-measure of ART compared to the one of random testing? To answer this question, we ran another set of experiments. For each of the 780 mutants discussed above, we considered set sizes ranging from 2 to 10 (reasonable values of test suite sizes that can be evaluated manually in practical conditions), and from 15 to 50 by intervals of 5 (only for sake of completeness, because for such short programs it would be unrealistic to have so many test cases evaluated by hand). This resulted in 17 set sizes, for a total of $13{,}260$ distinct experiments. For each of these experiments, we ran both random testing and ART 20 million times to get accurate estimates of the P-measure. Repeating the experiments 20 million times was necessary considering the fault rate values in our case study (Table 3). In total, we had $2 \times 20m \times 780 * \sum_{i=size} i \approx 9.7$ **tr**illion test case executions. Even with the use of a 200 node computer cluster, running these experiments took more than 10 days.

We calculated the effect size between the performance of random testing and ART using the *odds ratio* [22]. Given a, the number of times random testing finds a failure out of the $t = 20$ millions runs, and b the same number for ART, the odds ratio is calculated as $\psi = (a/t - a)/(b/t - b)$, where $\psi = 1$ means there is no effect size difference. Boxplots of the odds ratio for the $13{,}260$ experiments are shown in Figure 3 grouped by test set size. Fisher exact tests comparing a and b proportions at a 0.05 significance level show statistical significance in $12{,}530$ out of the $13{,}260$ experiments. P-measure boxplots for ART (i.e., b/t) are shown in Figure 4, reporting the median, 25% and 75% percentiles, and plot whiskers extend out from the boxes to the most extreme data point which is no more than 1.5 times the interquartile range from the box (this is the default setting in R [37]).

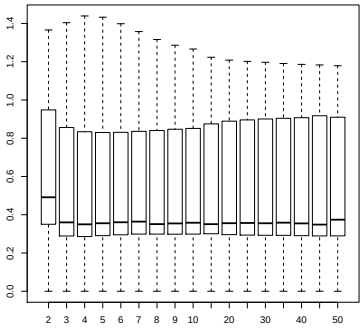

Figure 3: Odds ratios of random testing compared to ART on the 780 mutants and test sizes ranging from 2 to 50.

Although the results in Figure 3 suggest that ART can be better than random testing (the odds ratios are lower than 0.5 in most of cases), the results in Figure 4 show that ART is still very unlikely to detect faults: In most of the cases the P-measure is lower than 0.01, i.e., ART would have less than a 1% chance of finding failures. Even for large number of run test cases (i.e., 50), the performance is still unsatisfactory. Furthermore, such results are expected to be even worse on larger, more realistic software. In these cases, it could make more sense to use other testing strategies rather than ART, such as those based on structural coverage (e.g., branch coverage).

5. ANALYSIS OF THE ONE DIMENSION DOMAIN

This section aims at providing the necessary theoretical understanding to help explain some of the empirical results presented in Section 4. We will show that maximizing the distance between test cases, as performed by ART, may not necessarily, in all cases, improve fault detection rates. Furthermore, we will explain why ART is unlikely to be optimal in most cases. Though in order for the mathematical proofs to be tractable we base our demonstrations on simplifying assumptions (one dimension test input domain), we will discuss why the results are likely to generalize.

Let us consider an input domain D in which each test data is an

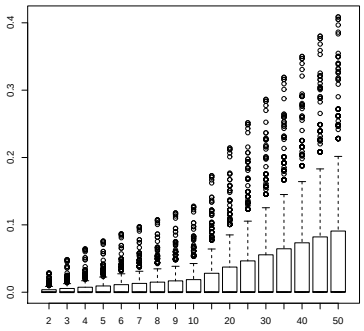

Figure 4: P-measures of ART on the 780 mutants and test sizes ranging from 2 to 50.

integer value in the range $[x_{min},x_{max}]$. Each value in that range belongs to the input domain, therefore $|D| = (x_{max} - x_{min}) + 1$. When there is one dimension ($n = 1$), the Euclidean distance of two integers is simply $dist(x,y) = |x - y|$.

For SUTs in which the input data is a single numerical variable, then the input domain would be a *line*. We have only one possible type of faulty region shape: one or more *segments* of total length $z = \theta|D|$, where $\theta = z/|D|$ is the failure rate. For a fixed size $|D|$ of the input domain, there would be only $(|D| - z) + 1$ positions in which a contiguous segment of length z can be placed. Let S_1 be the set of all possible SUTs in which a single integer is given as input, and for which a single faulty region of length z is present. It follows $|S_1| = (|D|-z)+1$. Note that empirical results [12] show that, for a constant failure rate, increasing the number of faulty regions (i.e., segments in one dimension) leads to lower ART detection rates compared to random testing.

Existing empirical studies [42, 1, 19, 38] provide evidence on the contiguity of faulty regions but not their positions in the input space. Therefore, we will assume in the absence of relevant information that all segments are equally likely.

Given K a set composed of $|K|$ test cases, let us consider it in ascending order of input data x, i.e., $x_i \leq x_{i+1}$ where $x_i,x_{i+1} \in K$. For example, we could have $K = \{2,5,7,13\}$. We can hence prove the following theorem:

Theorem 1. *For the set of programs S_1, assuming one faulty region of size z, a sufficient condition to maximize the probability $p_{S_1,K}$ that a set K of test cases detects a failure is when*

$$min(dist(x_{min},x_1),dist(x_i,x_{i+1}),dist(x_{|K|},x_{max})) \geq z\ . \quad (1)$$

PROOF. Given $\theta = z/|D|$ and $|S_1| = (|D| - z) + 1$, we start from studying on how many programs in S_1 a test data x can reveal a failure. Let x_{start} be the point for which the faulty region for a program $s \in S_1$ starts, i.e. for s we have the faulty region in $F_s = [x_{start},x_{start} + z - 1]$. A test data x reveals a failure in s if and only if $x \in F_s$.

Each program in S_1 has a different faulty region F, and this is defined by the $(|D| - z) + 1$ positions in which the starting point x_{start} can be positioned. Therefore, a test data x can reveal failures in at most z programs in S_1. This is an upper bound, because on the edges of D there would be less faulty regions. For example, if $x = x_{min}$, then there is only one program in S_1 for which x detects a failure, and that is when $x_{start} = x_{min}$.

An upper bound for $p_{S_1,K}$ would be $p_{S_1,K} \leq \frac{z|K|}{|S_1|} = \frac{|K|\theta}{(1-\theta)+\frac{1}{|D|}}$.

To prove this theorem, it is sufficient to prove that, when Condition 1 is met, then $p_{S_1,K} = \frac{z|K|}{|S_1|}$. To prove $p_{S_1,K} = \frac{z|K|}{|S_1|}$, we just need to prove that, when Condition 1 is met, (A) each test case in K reveals a failure in z programs in S_i and that (B) these programs are all different for each of the K test cases (so a total of $|K|z$ different failures are revealed).

Because from Condition 1,
$min(dist(x_{min},x_1),dist(x_{|K|},x_{max})) \geq z$,
then each test case in K finds failures in z programs, as the input data are far from the edges by at least a distance z, so $|F_s| = z$. This fulfills condition (A).

When $min(dist(x_i,x_{i+1})) \geq z$, to demonstrate (B), we need to prove that each program failure is triggered by exactly one test case. This can be proven by contradiction. Assume that x_i and x_j (where $i < j$) reveal failures in the same program s, hence $x_i,x_j \in F_s$. Because $|F_s| \leq z$, we have that $dist(x_i,x_j) = x_j - x_i < z$, but this is not possible considering that it should be $dis(x_i,x_j) = dist(x_i,x_{i+1}) + \sum_{z=i+1}^{j-1} dist(x_z,x_{z+1}) > dist(x_i,x_{i+1})$ but $dist(x_i,x_{i+1}) \geq z$, which contradicts Condition 1.

□

From Theorem 1, we can conclude that the higher the failure rate θ, the higher the threshold z beyond which increasing the distance between test cases does not help increase fault detection. There are two practical implications. First, maximizing the distance between test cases may not always be effective, at least not beyond a certain point. Second, when running a small number of random test cases in a large test input space, one is unlikely to violate the property in Theorem 1. Intuitively, considering the set of possible SUTs S_1, Theorem 1 can be explained by the fact that when selecting a new test case using ART, above a certain distance threshold from already selected test cases, the likelihood that this new test case will lie in a segment where existing test cases are already present is minimized. This threshold increases as the failure rate increases since there are more test cases that can fall in a given segment. At this stage we cannot mathematically prove that this holds for higher dimensions, but if this were to be confirmed in the more general case, this could explain why the difference in P-measure between random testing and ART can be small.

Though according to Theorem 1, we would not expect much difference between random testing and ART for small test set sizes, the results of the experiments in Figure 3 do not support this result: there can be large differences even for test set size 2 with odds ratios around 0.5. There is an explanation for that. Theorem 1 assumes that all the positions for the faulty regions have the same probability of occurrence, but the basic version of ART tends to sample data from the edges of the input space [12]. The results in Figure 3 can be hence explained if the faulty regions are more likely to be at the edges. Therefore, future work will need to study not only the shape of the faulty regions, but also their position in the input domain.

Based on the result of Theorem 1, it is possible to define a *deterministic* algorithm $\mathcal{DT}$ to choose $|K|$ test cases that maximize the probability of detecting failures for programs in S_1. The Java code for such an algorithm $\mathcal{DT}$ is shown if Figure 5. The main motivation for $\mathcal{DT}$ is not to devise a better solution than ART, as numerical programs with one input domain represent a small fractions of all programs, but rather to help compute an upper bound against which to compare ART in the one dimension case. $\mathcal{DT}$ simply chooses test cases that are as far as possible from each other and from the borders x_{min} and x_{max}. Given the range of values $|D| = (x_{max} - x_{min}) + 1$, we choose test cases that are at least $\delta = (|D| - 1)/(|K| + 1)$ from each other. Because the ratio δ is

```
public int[] get1DTestCases(int k, int min, int max){
  int[] K = new int[k];
  int range =  (max - min);
  int delta = (int)Math.floor((double)range/(double)(k+1));
  int r = range - delta*(k+1);
  K[0] = min + delta;
  if(r>0){ K[0]++; r--;}
  for(int i=1; i < K.length; i++){
    K[i] = K[i-1]+delta;
    if(r>0){ K[i]++; r--;}
  }
  return K;
}
```

Figure 5: Java code of the deterministic algorithm $\mathcal{DT}$ for the set S_1.

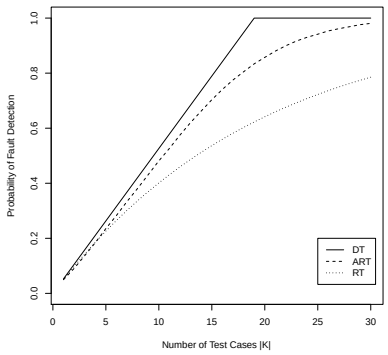

Figure 6: Comparison of ART with $\mathcal{DT}$ and random testing on S_1.

not necessarily an integer value, the algorithm is designed to handle these situations. The optimality of $\mathcal{DT}$ is proven in the following theorem. The proof is based on the Java code listed in Figure 5, but it would apply on any equivalent implementation in other programming languages. An integer array K is given as output. Notice that K[j] represents the input data x_{j+1} (this because Java arrays start from index 0).

Theorem 2. *For the set of programs S_1 and any number of test cases $|K|$ to sample, the algorithm $\mathcal{DT}$ produces a set K of test cases that maximizes the probability of detecting failures in S_1.*

PROOF. Given K test cases as output of $\mathcal{DT}$, if Condition 1 of Theorem 1 holds, then this theorem is true as the probability of fault detection is maximized. Condition 1 is however sufficient but not necessary. When Condition 1 does not hold for the K test cases sampled by $\mathcal{DT}$, to prove Theorem 2, it is sufficient to prove that $p_{S_1,K} = 1$, implying that the probability of detecting failures is still maximized.

In the code of $\mathcal{DT}$, we have the variable $\delta = \lfloor (|D|-1)/(|K|+1) \rfloor$. For each test case, we have that $x_i \geq x_{i-1}$, because x_i is calculated by adding δ to x_{i-1}. In some cases, we have $x_i = x_{i-1} + \delta + 1$. Therefore, for the sampled test cases we have $\delta \leq dist(x_i,x_{i+1}) \leq \delta + 1$.

For x_1 we used the value $x_{min} + \delta$ to which $+1$ can be added in some cases. Therefore, x_1 is at most $\delta + 1$ values far from the edge x_{min}, i.e. $\delta \leq dist(x_{min},x_1) \leq \delta + 1$.

The distance of $x_{|K|}$ from the edge x_{max} needs some more steps. We have $x_{|K|} = x_1 + \sum_{i=2}^{|K|}(x_i - x_{i-1})$ to which can be added $+1$. This addition is based on the counter r which starts from the value $r = (|D|-1) - \delta(|K|+1)$. We hence have $x_{|K|} = x_{min} + \delta|K| + r = x_{min} + |D| - 1 - \delta$. Therefore, $dist(x_{|K|},x_{max}) = x_{max} - x_{min} - |D| + 1 + \delta = \delta$.

If the conditions of Theorem 1 do not hold, this means that there is at least one data input x_i that is at least within a distance z from another input data or from the edges of the input domain. Because we have proven that minimum distance should be at least δ, then $z > \delta$, otherwise the conditions of Theorem 1 would hold. From $z > \delta$ it simply follows that $z \geq \delta + 1$. Because we have proven that $\delta + 1$ is the maximum distance between two consecutive x_i and x_{i+1}, and that x_1 and $x_{|K|}$ are within that distance from the edges, then K finds failures in each program in S_1. This is because $z \geq \delta + 1$. Therefore, in these conditions we have $p_{S_1,K} = 1$.

□

Considering that the algorithm $\mathcal{DT}$ is optimal (Theorem 2) and deterministic, the application of ART on SUTs with one dimension domains would be questionable. However, $\mathcal{DT}$ requires that the number $|K|$ of test cases to sample should be decided before starting the algorithm. This is acceptable in many practical situations when no automated oracle is available or test cases are expensive as they are run, for example, on embedded software with actual hardware. In such cases only a limited number $|K|$ of test cases can be executed. Similarly, in regression testing [31], only a relatively small subset of regression test cases can be selected in many situations. On the other hand, in other situations than the ones described above, it could make sense to run a testing technique until it detects the first failure. $\mathcal{DT}$ is, however, optimal when *all* the sampled test cases in K are executed. If the cost of running the test cases is high and/or K is large, then it may be desirable to order their execution using prioritization techniques [28].

We evaluated ART on the set S_1 and compared it with random testing using $\mathcal{DT}$ as an upper bound. To do so, we carried out a simulation in which $|D| = 10{,}000$, $\theta = 0.05$ and where we considered sets of test cases ranging from 1 to 30. Notice that, given $\theta = 0.05$, then on average random testing would require to sample 20 test cases to find a failure (this is a very high failure rate, used only for the sake of illustration). Figure 6 compares the fault detection capabilities of these three techniques. When a testing technique outputs a set of K test cases, we evaluate its fault detection capability by running it on *all* the SUTs in S_1 for the given values of $|D|$ and θ, so $|S_1| = (|D| - z) + 1 = 9{,}501$. The probability of fault detection is estimated by dividing the number of programs in S_1 for which a failure is revealed by the total number of programs $|S_1|$. Since $\mathcal{DT}$ is a deterministic algorithm, we only needed to evaluate it once on each program in S_1. On the other hand, because ART is randomized, we ran it on each program in S_1 1000 times, and report the average in Figure 6. The fault detection probability of random testing is simply calculated with the following formula: $p = 1 - (1-\theta)^{|K|}$ [18].

As we can see in Figure 6, ART seems better than random testing, but it is far from the optimal $\mathcal{DT}$. The difference between these techniques increases as the number of test cases increases and is negligible for small test set sizes. This is explained by Theorem 1 as there is an increasing number of pairs of test cases whose distance is below the z, the length of the segment (faulty region).

Is it possible to define a new variant of ART to get closer to the performance of $\mathcal{DT}$? This would be useful in the cases in which an automated oracle is available, because we could run ART until it finds the first failure, instead of having to specify $|K|$ as in $\mathcal{DT}$.

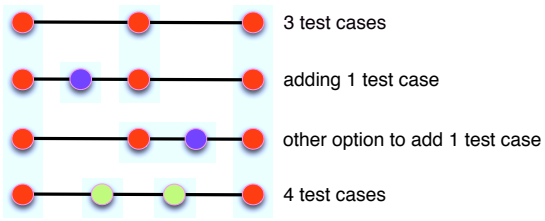

Figure 7: Graphical example of test cases (circles) for S_1 (line).

Unfortunately, this is not possible for techniques in which test cases are chosen one at the time (stepwise, greedy) as in ART. Figure 7 shows an example to illustrate this problem. Assume that we have three test cases that should be as diverse as possible. The optimal allocation would be one in the centre and the other two on the edges (notice that choosing test cases at the edges is not a good idea [8], and $\mathcal{DT}$ does not do it). If then we want to add one test case, we have two options to where to allocate it if we want to maximize diversity. But in both cases, the allocation of these four test cases is not the optimal one that maximizes diversity (as shown in the last row of Figure 7).

In *antirandom* testing [43] diversity in the test cases is sought, in a similar way as in ART. But each time a new test case is sampled, the choice is made deterministically. In contrast to $\mathcal{DT}$, antirandom testing is affected by the problem we discussed for Figure 7.

For higher dimensions, we cannot at the moment derive a precise $\mathcal{DT}$ algorithm, as it depends on the many possible shapes of faulty regions, and it is therefore impossible to compare ART with an optimal solution for more than one dimension at this point. But given the reasons we have presented to explain why ART is suboptimal in the one dimensional case, we can conjecture that ART is likely to be suboptimal in the general case as well. It is very well possible that if we manage to modify ART to better optimize diversity, we might obtain better empirical results than the ones presented in the previous section.

6. GENERALIZATION

Though we only discussed ART for numerical applications (the type of SUT that ART was originally designed for), the reasons we identified to explain ART inefficiencies (e.g., distance computations and memory consumption) can be generalized to other types of programs. In situations where one is testing until triggering a failure, any situation in any domain when the execution time of test cases is not high compared to distance computations, or when the failure rate is low, will lead to a low ART performance. Reported applications of ART to non-numerical programs is further discussed in the next section. Alternatively, when a fixed, small size test set is used (e.g., due to manual oracles) then ART will still be unlikely to detect faults in any situation, regardless of the domain, where the failure rate is low as the ART heuristic will not be sufficiently effective to significantly improve over random testing, as discussed in Section 4.3. More empirical studies are however required to generalize our results with confidence.

The only case left where ART could be cost-effective is when there is an automated oracle (so that the F-measure makes sense) and text execution time is on average extremely long compared to distance computation time. Formalizing this trade-off with an inequality yields:

$$E[ART] \times t_c + \#DC \times t_d \leq E[RT] \times t_c \ ,$$

where $E[RT] = 1/\theta$ is the expected number of test cases sampled by random testing, $E[ART] = 1/2\theta$ is the expected number of test cases sampled by ART (in the best case in which it only requires to sample half test cases than random testing), $\#DC = |W|\frac{1}{2\theta}(\frac{1}{2\theta} - 1)/2$ is the number of distance calculations (recall W is the number of test cases sampled at each step of the ART algorithm), t_c is the time to execute a test case, and t_d is the time it takes to compute a distance value. Notice that this inequality does not take into account the memory overhead. This inequality yields:

$$t_c \geq \frac{|W|}{2} \times (\frac{1}{2\theta} - 1) \times t_d \ . \quad (2)$$

From Inequality 2, we can see that, for example using a failure rate of $\theta = 10^{-5}$, test execution would need to take on average 2.5×10^5 more time than a distance computation. If we consider a distance computation to take one millisecond, this will result in more than four minutes of execution per test case. What we can conclude from the above inequality and example, especially given that this is the best case scenario for ART, is that situations where test case execution is long enough to warrant the use of ART are likely to be rare in practice. Furthermore, the above reasoning does not account for memory consumption which, using the example above, would result in many megabytes of test case information. Though extensions of the basic ART exist [11, 32, 26, 6], they have never been empirically investigated in realistic scenarios (as discussed in more details in Section 4).

7. ART IN OTHER TESTING DOMAINS

In previous sections, we carried out theoretical and empirical analyses that investigated the effectiveness of ART in numerical applications and showed we were unlikely to find many situations where it could be expected to be satisfactory. In this section, we analyze the published literature applying ART to other domains and conclude that existing results cast further doubts about its effectiveness.

Ciupa *et al.* [15] used ART for testing Object-Oriented software, in which "contracts" (i.e., pre/post conditions in the tested methods) were used as automated oracles. The authors concluded that ART was better than random testing since it needed to sample less test cases before finding the first failure. However, ART was also reported as taking on average 1.6 *times longer* due to the distance calculations! As we argued above, when you have an automated oracle, how many test cases you sample is simply irrelevant: the time required to execute the test cases is the only important metric in this case. The number of test cases would be important only if the outputs of the test cases needed some form of manual checks. Therefore, in contrast to what it is claimed in [15] , the empirical analysis in that paper actually shows that ART fares significantly worse than random testing.

Lin *et al.* [29] also analyzed ART on Object-Oriented software and compared results using execution time instead of the F-measure. However, the faults in that work were *manually* seeded, which led to failure rates that were extremely high (in the order of 10^{-3}). As in our study earlier, it is important to consider a wide and realistic range of failure rates, especially if the objective of the study is to demonstrate applicability. Furthermore, in both [15] and [29] the experiments were repeated only five times and, as a result, no statistical testing was used to assess whether differences in execution time of F-measure were statistically significant. When randomized algorithms are evaluated, since there can be high variance in the performance at each run, using higher number of repeated experiments and statistical tests is a necessity [3].

In our recent work with Iqbal [4], we applied ART for system testing of a real industrial embedded system with real faults. This was a priori an ideal scenario for ART: computationally expensive test cases (each one required 20 seconds), automated oracles (provided by the environment models), and high failure rate due to the early testing stage of the studied industrial system. However, even in such conditions, the results showed that there was no statistical difference between the performance of ART and random testing.

In our work with Hemmati [24], we applied ART to select subsets of test cases generated using state machine models of an industrial software. Our test strategy yielded 281 test cases in order to achieve adequate coverage of the state machine. Because running so many system test cases was infeasible due to the use of actual hardware and network infrastructure, the goal was to study strategies for selecting significantly smaller subsets of test cases. Though rewarding diversity in the selected subset of test cases (which can be considered to be an optimization problem) led to better fault detection than random selection, ART was statistically worse than the other algorithms we investigated, such as Genetic Algorithms.

Jiang *et al.* [25] and Zhou [44] used ART for prioritizing test cases in regression testing. Rewarding diversity using ART led to better results. However, in contrast to our previous work [24], the use of optimization algorithms (e.g., Genetic Algorithms) for rewarding diversity was not investigated. We cannot claim that such algorithms would be better than ART in the case studies used in [25] and [44], but we can make the following conjecture. ART works as a *greedy algorithm*, and rewarding diversity in a set of test cases can be considered to be an optimization problem. In general, meta-heuristic techniques (e.g., Genetic Algorithms) tend to yield better results than greedy algorithms for non-trivial problems [21]. This was true in our case study in [24], and can be expected to be so in many software engineering applications [23].

Notice that for the types of problem in [25, 44, 24] it is not possible to use a deterministic algorithm such as $\mathcal{DT}$. The reason is that, in contrast to numerical applications, in those cases we cannot in general directly produce test cases with a pre-defined distance among them. We can only sample test cases and then, only after they are created, measuring their distance becomes possible.

The above analysis of the literature applying ART to non-numerical applications confirms that ART is not a particularly effective testing technique in most cases. Diversity in test cases is an intuitive heuristic, but ART may not be an advantageous option to achieve it.

8. CONCLUSION

Adaptive Random Testing (ART) was proposed as an improvement to random testing, with many studies carried out over the years. ART was first proposed for numerical applications [13], and was more recently applied to other testing problems such as testing Object-Oriented classes [15]. Its underlying principle is intuitive and appealing: maximizing test case diversity.

In this paper, we revisited the claims made in many ART studies and we showed that, using similar programs in the numerical application domain, ART does not work as well as expected. We further explain why ART does not work well in two practical and common situations where oracles are automated and one tests until a failure is triggered or when small test suites of fixed size are executed and the oracles manually checked. In the former case *ART does not even work on toy problems such as Triangle Classification!* We identified precise reasons of this behavior, which are not specific to our case studies and are therefore likely to generalize to other domains under certain conditions: short test execution times relative to distance calculations and low failure rates. One major reason for this is the calculation of distances among test cases, which overshadow the reduction that ART could yield in terms of number of executed test cases, especially under the conditions mentioned above. With small test suites of fixed size, ART improves over random testing but not sufficiently so to make a practical difference, especially when failure rates are realistic.

The results above were unexpected if we consider the large amount of research that has been carried out over the years on ART (Table 5) and the claims that were made in these publications. However, a closer look at the studies reported in the literature showed serious flaws, which are rooted in unclear and unrealistic assumptions (e.g., very high failure rates) and inadequate measurement (e.g., F-measure). The lack of cost-effectiveness of ART seems present also in the other testing domains in which ART has been applied.

On the other hand, in this paper we have also provided further empirical evidence to support the contiguous faulty region hypothesis underlying the ART heuristic. We propose ways to automate its investigation in large scale studies by using metrics characterizing the shape of faulty regions. Future work will be devoted to using the findings of this paper to design novel testing techniques that can more effectively exploit the faulty region hypothesis than current versions of ART.

Acknowledgements

We would like to thank Johannes Mayer for providing the Java source code of his case study used in [38]. We also like to thanks Tsong Chen and Lydie du Bousquet for useful comments on an early draft of this paper. The work described in this paper was supported by the Norwegian Research Council. This paper was produced as part of the ITEA-2 project called VERDE.

9. REFERENCES

[1] P. E. Ammann and J. C. Knight. Data diversity: An approach to software fault tolerance. *IEEE Transations on Computers*, 37(4):418–425, 1988.

[2] J. H. Andrews, L. C. Briand, Y. Labiche, and A. S. Namin. Using mutation analysis for assessing and comparing testing coverage criteria. *IEEE Transactions on Software Engineering (TSE)*, 32(8):608–624, 2006.

[3] A. Arcuri and L. Briand. A practical guide for using statistical tests to assess randomized algorithms in software engineering. In *ACM/IEEE International Conference on Software Engineering (ICSE)*, 2011.

[4] A. Arcuri, M. Z. Iqbal, and L. Briand. Black-box system testing of real-time embedded systems using random and search-based testing. In *IFIP International Conference on Testing Software and Systems (ICTSS)*, pages 95–110, 2010.

[5] A. Arcuri, M. Z. Iqbal, and L. Briand. Formal analysis of the effectiveness and predictability of random testing. In *ACM International Symposium on Software Testing and Analysis (ISSTA)*, pages 219–229, 2010.

[6] T. Y. Chen, D. Huang, F. Kuo, R. Merkel, and J. Mayer. Enhanced lattice-based adaptive random testing. In *Proceedings of the ACM symposium on Applied Computing*, pages 422–429, 2009.

[7] T. Y. Chen, F. Kuo, and Z. Zhou. On favourable conditions for adaptive random testing. *International Journal of Software Engineering and Knowledge Engineering*, 17(6):805–825, 2007.

[8] T. Y. Chen, F. C. Kuo, and H. Liu. Distributing test cases more evenly in adaptive random testing. *Journal of Systems and Software (JSS)*, 81(12):2146–2162, 2008.

[9] T. Y. Chen, F. C. Kuo, and H. Liu. Adaptive random testing based on distribution metrics. *Journal of Systems and Software (JSS)*, In Press:–, 2009.

[10] T. Y. Chen, F. C. Kuo, and H. Liu. Application of a failure driven test profile in random testing. *IEEE Transactions on Reliability*, 58(1):179–192, 2009.

[11] T. Y. Chen, F. C. Kuo, R. G. Merkel, and S. Ng. Mirror adaptive random testing. *Information and Software Technology (IST)*, 46(15):1001–1010, 2004.

[12] T. Y. Chen, F. Kuoa, R. G. Merkela, and T. Tseb. Adaptive random testing: The art of test case diversity. *Journal of Systems and Software (JSS)*, 2010. (in press).

[13] T. Y. Chen, H. Leung, and I. K. Mak. Adaptive random testing. In *Advances in Computer Science*, pages 320–329, 2004.

[14] T. Y. Chen and R. Merkel. An upper bound on software testing effectiveness. 17(3):1–27, 2008.

[15] I. Ciupa, A. Leitner, M. Oriol, and B. Meyer. Artoo: adaptive random testing for object-oriented software. In *ACM/IEEE International Conference on Software Engineering (ICSE)*, pages 71–80, 2008.

[16] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms.* MIT Press and McGraw-Hill, second edition, 2001.

[17] J. W. Duran and S. C. Ntafos. An evaluation of random testing. *IEEE Transactions on Software Engineering (TSE)*, 10(4):438–444, 1984.

[18] W. Feller. *An Introduction to Probability Theory and Its Applications, Vol. 1*. Wiley, 3 edition, 1968.

[19] G. B. Finelli. Nasa software failure characterization experiments. *Reliability Engineering & System Safety*, 32:155–169, 1991.

[20] P. Godefroid, N. Klarlund, and K. Sen. Dart: directed automated random testing. In *ACM Conference on Programming language design and implementation (PLDI)*, pages 213–223, 2005.

[21] D. E. Goldberg. *Genetic Algorithms in Search and Optimization*. Addison-wesley, 1989.

[22] R. Grissom and J. Kim. *Effect sizes for research: A broad practical approach*. Lawrence Erlbaum, 2005.

[23] M. Harman, S. A. Mansouri, and Y. Zhang. Search based software engineering: A comprehensive analysis and review of trends techniques and applications. Technical Report TR-09-03, King's College, 2009.

[24] H. Hemmati, A. Arcuri, and L. Briand. Reducing the cost of model-based testing through test case diversity. In *IFIP International Conference on Testing Software and Systems (ICTSS)*, 2010.

[25] B. Jiang, Z. Zhang, W. Chan, and T. Tse. Adaptive random test case prioritization. In *IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 233–244, 2009.

[26] F. C. Kuo. An indepth study of mirror adaptive random testing. In *International Conference on Quality Software*, pages 51–58, 2009.

[27] F. C. Kuo, T. Y. Chen, H. Liu, and W. K. Chan. Enhancing adaptive random testing for programs with high dimensional input domains or failure-unrelated parameters. *Software Quality Journal*, 16(3):303–327, 2008.

[28] Z. Li, M. Harman, and R. M. Hierons. Meta-heuristic search algorithms for regression test case prioritization. *IEEE Transactions on Software Engineering (TSE)*, 33(4):225–237, 2007.

[29] Y. Lin, X. Tang, Y. Chen, and J. Zhao. A divergence-oriented approach to adaptive random testing of java programs. In *IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 221–232, 2009.

[30] Y. S. Ma, J. Offutt, and Y. R. Kwon. Mujava: an automated class mutation system. *Software Testing, Verification and Reliability*, 15(2):97–133, 2005.

[31] N. Mansour, R. Bahsoon, and G. Baradhi. Empirical comparison of regression test selection algorithms. *Journal of Systems and Software*, 57(1):79–90, 2001.

[32] J. Mayer. Lattice-based adaptive random testing. In *IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 333–336, 2005.

[33] J. Mayer and C. Schneckenburger. An empirical analysis and comparison of random testing techniques. In *ACM/IEEE International symposium on Empirical Software Engineering*, pages 105–114, 2006.

[34] G. Myers. *The Art of Software Testing*. Wiley, New York, 1979.

[35] C. Pacheco, S. K. Lahiri, M. D. Ernst, and T. Ball. Feedback-directed random test generation. In *ACM/IEEE International Conference on Software Engineering (ICSE)*, pages 75–84, 2007.

[36] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical recipes: the art of scientific computing*. Cambridge University Press, 2007.

[37] R Development Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2008. ISBN 3-900051-07-0.

[38] C. Schneckenburger and J. Mayer. Towards the determination of typical failure patterns. In *International workshop on Software quality assurance*, pages 90–93, 2007.

[39] S. Shin, S. Park, K. Choi, and K. Jung. Normalized Adaptive Random Test for Integration Tests. In *IEEE International Workshop on Software Test Automation*, 2010.

[40] A. F. Tappenden and J. Miller. A novel evolutionary approach for adaptive random testing. *IEEE Transactions On Reliability*, 58(4):619–633, 2009.

[41] T. A. Thayer, M. Lipow, and E. C. Nelson. *Software Reliability*. North Holland, Amsterdam, 1978.

[42] L. J. White and E. I. Cohen. A domain strategy for computer program testing. *IEEE Transactions on Software Engineering (TSE)*, 6(3):247–257, 1980.

[43] S. H. Wu, S. Jandhyala., Y. K. Malaiya, and A. P. Jayasumana. Antirandom testing: a distance-based approach. *VLSI Design*, 10:1–9, 2008.

[44] Z. Zhou. Using Coverage Information to Guide Test Case Selection in Adaptive Random Testing. In *IEEE International Workshop on Software Test Automation*, 2010.