Adaptive Random Testing

T.Y. Chen[1,⋆], H. Leung[2], and I.K. Mak[3]

[1] School of Information Technology, Swinburne University of Technology, Hawthorn, Victoria 3122, Australia
tchen@it.swin.edu.au
[2] Department of Computer Science, New Mexico State University, Las Cruces, NM 88003, U.S.A.
hleung@nmsu.edu
[3] School of Professional and Continuing Education, The University of Hong Kong, Pokfulam Road, Hong Kong
keith.mak@hkuspace.hku.hk

Abstract. In this paper, we introduce an enhanced form of random testing called *Adaptive Random Testing*. Adaptive random testing seeks to distribute test cases more evenly within the input space. It is based on the intuition that for non-point types of failure patterns, an even spread of test cases is more likely to detect failures using fewer test cases than ordinary random testing. Experiments are performed using published programs. Results show that adaptive random testing does outperform ordinary random testing significantly (by up to as much as 50%) for the set of programs under study. These results are very encouraging, providing evidences that our intuition is likely to be useful in improving the effectiveness of random testing.

1 Introduction

There are basically two approaches towards the selection of test cases, namely the white box and the black box approach. Among the black box techniques, random selection of test cases is generally regarded as not only a simple but also an intuitively appealing technique (e.g. see White [1]). In random testing, test cases may be randomly chosen based on a uniform distribution or according to the operational profile. As pointed out by Hamlet [2], the main merits of random testing include the availability of efficient algorithms to generate its test cases, and its ability to infer reliability and statistical estimates.

In all random testing studies, only the rate of failure-causing inputs (hereafter referred to as the failure rates) is used in the measurement of effectiveness. For example, the expected number of failures detected and the probability of detecting at least one failure are all defined as functions of the failure rates. However, in a recent study by Chan et al. [3], it has been found that the performance of a partition testing strategy depends not only on the failure rate, but also on the

⋆ Corresponding author.

M.J. Maher (Ed.): ASIAN 2004, LNCS 3321, pp. 320–329, 2004.

geometric pattern of the failure-causing inputs. This has prompted us to investigate whether the performance of random testing can be improved by taking the patterns of failure-causing inputs into consideration. We have developed a new type of random testing, namely adaptive random testing. Our studies show that the effectiveness of random testing can be significantly improved without incurring significant overheads. We observe that adaptive random testing outperforms ordinary random testing in general.

2 Preliminaries

In this study, we assume that the random selection of test cases is based on a uniform distribution and without replacement. As a reminder, most of the analytical studies of random testing assume that selections are with replacement [4]. The assumption of selection with replacement has long been criticised by practitioners, because in reality, test cases should not be repeated. It has been used mainly because it has a simpler mathematical model and hence of easier analysis. Thus, the model of our study reflects the reality more closely.

We basically follow the notation used by Chen and Yu [4]. Elements of an input domain are known as failure-causing inputs, if they produce incorrect outputs. For an input domain D, we use d, m and n to denote the size, number of failure-causing inputs and number of test cases, respectively. The sampling rate σ and failure rate θ are defined as $\frac{n}{d}$ and $\frac{m}{d}$, respectively.

In previous studies of random testing, the two most popular effectiveness metrics are: the probability of detecting at least one failure (referred to as the P-measure) and the expected number of failures detected (referred to as the E-measure). There have been some criticisms on these two metrics despite their popularity. The main criticism of using the E-measure is that higher E-measures do not necessarily imply more faults or more distinct failures; and the main disadvantage of using the P-measure is that there is no distinction between cases of detecting different number of failures. Although these two measures are not ideal, they have been used extensively in the literature.

In this paper, rather than using the P-measure and the E-measure as the effectiveness metrics as done in previous studies, we propose another effectiveness metric. We use the expected number of test cases required to detect the first failure (referred to as the F-measure), as the effectiveness metric. For random selection of test cases with replacement, the F-measure, denoted by F, is equal to $\frac{1}{\theta}$, or equivalently $\frac{d}{m}$. Intuitively speaking, the F-measure reflects the effectiveness of a testing strategy more naturally and directly, as the lower the F-measure the more effective the testing strategy because fewer test cases are required to reveal the first failure. In practice, when a failure is detected, testing is normally stopped and debugging starts. The testing phase would normally be resumed only after fixing of the fault. Hence, the F-measure is not only more intuitively appealing but also more realistic from a practical perspective.

In a recent study, Chan et al. [3] have observed that the performance of some partition testing strategies varies with the patterns of failure-causing inputs (hereafter referred to as the failure patterns). They have classified the patterns of failure-causing inputs into three categories: point, strip and block patterns. To illustrate this, let us assume that the input domain has two dimensions. Figures 1-3 show the point, strip and block patterns, respectively. The outer boundaries represent the borders of the input domain and the filled regions denote the failure-causing inputs, that is the failure patterns, in each of the figures.

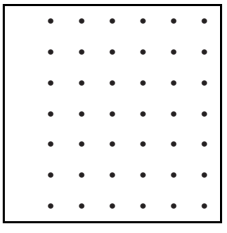

Fig. 1. Point Pattern

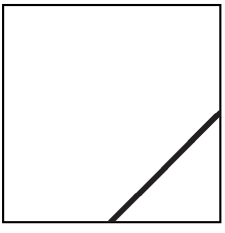

Fig. 2. Strip Pattern

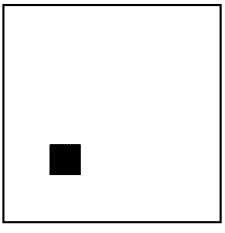

Fig. 3. Block Pattern

The main characteristic of the point pattern is that either the failure-causing inputs are stand alone points, or they form regions of a very small size which are scattered over the whole domain. For the strip pattern, the failure-causing inputs form the shape of a narrow strip. A typical example of this failure pattern is White and Cohen's [5] domain errors. For the block pattern, the main characteristic is that the failure-causing inputs are concentrated in either one or a few regions.

3 Adaptive Random Testing

With ordinary random testing, the chances of hitting the failure patterns, that is selecting failure-causing inputs as test cases, depends solely on the magnitude of the failure rate. However, a closer inspection shows that for non-point patterns which include both the strip and block patterns, the failure detection capability can be significantly improved by slightly modifying the ordinary random testing technique. Let us use an example to illustrate the intuition behind our modified random testing.

Consider an input domain D. Suppose D has a "regular" geometry, that is, we assume that it is easy to generate inputs randomly from D. Let there be a fault in the program. For example, let the input in D be consisting of values x and y where $0 \leq x, y \leq 10$. Suppose the fault lies in a conditional expression

$$x + y > 3$$

at a program statement while the correct expression should have been

$$x + y > 4$$

Specifically, the domain D is a square $\{(x, y) \mid 0 \leq x, y \leq 10\}$. The fault corresponds to a failure region $\{(x, y) \mid 3 < x + y \leq 4\}$, which is a "strip" of width 1 across the square domain D.

We consider applying random testing to the program. Suppose we generate one test case (2.2,2.2). Since 2.2+2.2 is larger than 4, the fault is not revealed. Let the next test case generated be (2.1,2.1). Should we know in advance that the error region is a strip of width 1, we could argue that the choice of the second test case is too "humble". The test set should better be more spaced out such that two neighbouring test cases are kept apart by a distance of length at least 1.

Thus, we propose to modify random testing as follows. To generate a new test case, we need to make sure that the new test case should not be too close to any of the previously generated ones. One way to achieve this is to generate a number of random test cases and then choose the "best" one among them. That is, we try to distribute the selected test cases as spaced out as possible.

To summarize, we conjecture that *test cases should be as evenly spread over the entire input domain as possible* in order to achieve a small F-measure. This forms the basis of our new method of random testing, namely *Adaptive Random Testing*.

An implementation of *Adaptive Random Testing* is as follows: Adaptive random testing makes use of two sets of test cases, namely the executed set and the candidate set which are disjoint. The executed set is the set of distinct test cases that have been executed but without revealing any failure; while the candidate set is a set of test cases that are randomly selected without replacement. The executed set is initially empty and the first test case is randomly chosen from the input domain. The executed set is then incrementally updated with the selected element from the candidate set until a failure is revealed. From the candidate set, an element that is *farthest away* from all executed test cases, is selected as the next test case.

Obviously, there are various approaches to implement the intuition of "*farthest away*". In this paper the criterion for implementing this intuition is defined as follows. Let $T = \{t_1, t_2, \cdots, t_n\}$ be the executed set and $C = \{c_1, c_2, \cdots, c_k\}$ be the candidate set such that $C \cap T = \emptyset$. The criterion is to choose the element c_h such that for all $j \in \{1, 2, \cdots, k\}$,

$$\min_{i=1}^{n} dist(c_h, t_i) \geq \min_{i=1}^{n} dist(c_j, t_i)$$

where *dist* is defined as the Euclidean distance.

In other words, in an m-dimensional input domain, for inputs $a = (a_1, a_2, \cdots, a_m)$ and $b = (b_1, b_2, \cdots, b_m)$, $dist(a, b) = \sqrt{\sum_{i=1}^{n} (a_i - b_i)^2}$. The rationale of this criterion is to evenly spread the test case through maximising the minimum distance between the next test case and the already executed test cases.

It should be noted that there are also various ways to construct the candidate set giving rise to various versions of adaptive random testing. It will be quite difficult, if not impossible, to carry out an analytical study for the performance

Table 1. Program name, dimension (D), input domain, seeded error types, and total number of errors for each of the error-seeded programs. The error types are: arithmetic operator replacement (AOR); relational operator replacement (ROR); scalar variable replacement (SVR) and constant replacement (CR)

Prog		Input Domain		Error Type				Total	Failure
Name	D	From	To	AOR	ROR	SVR	CR	Errors	Rate
airy	1	(−5000.0)	(5000.0)				1	4	0.000716
bessj	2	(2.0, −1000.0)	(300.0, 15000.0)	2	1		1	4	0.001298
bessj0	1	(−300000.0)	(300000.0)	2	1	1	1	5	0.001373
cel	4	(0.001, 0.001, 0.001, 0.001)	(1.0, 300.0, 10000.0, 1000.0)	1	1		1	3	0.000332
el2	4	(0.0, 0.0, 0.0, 0.0)	(250.0, 250.0, 250.0, 250.0)	1	3	2	3	9	0.000690
erfcc	1	(−30000.0)	(30000.0)	1	1	1	1	4	0.000574
gammq	2	(0.0, 0.0)	(1700.0, 40.0)		3		1	4	0.000830
golden	3	(−100.0, −100.0, −100.0)	(60.0, 60.0, 60.0)		3	1	1	5	0.000550
plgndr	3	(10.0, 0.0, 0.0)	(500.0, 11.0, 1.0)	1	2		2	5	0.000368
probks	1	(−50000.0)	(50000.0)	1	1	1	1	4	0.000387
sncndn	2	(−5000.0, −5000.0)	(5000.0, 5000.0)			4	1	5	0.001623
tanh	1	(−500.0)	(500.0)	1	1	1	1	4	0.001817

of the adaptive random testing in comparison with the ordinary random testing with respect to the F-measure. Therefore, we conduct an empirical study in this paper.

4 An Empirical Study on Adaptive Random Testing

In this section, an empirical investigation was conducted to compare the performance between adaptive random testing and ordinary random testing, using the F-measure as the effectiveness metric, which is defined as the expected number of test cases required to detect the first failure. In this paper, F_a and F_r are used to denote the F-measures for the adaptive random testing and ordinary random testing, respectively. Unless otherwise specified, ordinary random testing would be abbreviated as random testing or RT.

In this empirical study, we use a set of 12 error-seeded programs. They are all published programs ([6, 7]), which are written in Fortran, Pascal or C with program sizes ranging from 30 to 200 statements. All of them involve numerical computations and have been converted into C++ programs. Table 1 lists details of the failure rate, type and number of seeded errors for each program.

The candidate set is of constant size and a new candidate set is constructed each time a test case is selected. Let us refer it as the *Fixed Size Candidate Set Version of the Adaptive Random Testing* (abbreviated as the FSCS). Algorithm 1 describes how to generate a candidate set and select a test case for this version of adaptive random testing.

Firstly, we conducted a preliminary study to see how the size of the candidate set would affect the performance of the adaptive random testing for our sample of 12 programs. The range of the size of the candidate set, k, was set to vary from 2 to 10 with an increment of 2 and then from 10 to 100 with an increment of 10. We have observed that in general, the larger the k, the smaller the number of test cases required to detect the first failure. Furthermore, for $k \geq 10$, there not much difference in the number of test cases required to detect the first failure in our sample of programs. Hence, we set $k = 10$ in our experiment.

```
Algorithm 1:
/*
selected_set := { test data already selected };
candidate_set := {};
total_number_of_candidates := 10;
*/
function Select_The_Best_Test_Data(selected_set, candidate_set,
                    total_number_of_candidates);
    best_distance := −1.0;
    for i := 1 to total_number_of_candidates do
        candidate := randomly generate one test data from the program
                    input domain, the test data cannot be in
                    candidate_set nor in selected_set;
        candidate_set := candidate_set + { candidate };
        min_candidate_distance := Max_Integer;
        foreach j in selected_set do
            min_candidate_distance := Minimum(min_candidate_distance,
                Euclidean_Distance(j, candidate));
        end_foreach
        if (best_distance < min_candidate_distance) then
            best_data := candidate;
            best_distance := min_candidate_distance;
        end_if
    end_for
    return best_data;
end_function
```

For each program, we applied both FSCS and RT with the same first randomly selected test case. We obtained a pair of numbers (u^a, u^r), where u^a and u^r were the numbers of test cases required to detect the first failure, using FSCS (Algorithm 2) and RT, respectively. We called such a pair of numbers a sample. Obviously, (u^a, u^r) depends on the first test case. Hence, the process was repeated with various randomly selected inputs as the first test case. The central limit theorem [8] was used to determine the size of the sample set $S = \{(u_1^a, u_1^r), (u_2^a, u_2^r), \cdots\}$, that is, to determine when the process could be stopped so that we have enough samples in S to provide reliable statistic estimates.

```
Algorithm 2:
initial_test_data := randomly generate a test data from the input domain;
selected_set := { initial_test_data };
counter := 1;
total_number_of_candidates := 10;
use initial_test_data to test the program;
if (program output is incorrect) then
        reveal_failure := true;
else
        reveal_failure := false;
end_if
while (not reveal_failure) do
        candidate_set := {};
        test_data := Select_The_Best_Test_Data(selected_set, candidate_set,
                                total_number_of_candidates);
        use test_data to test the program;
        if (program output is incorrect) then
                reveal_failure := true;
        else
                selected_set := selected_set + { test_data };
                counter := counter + 1;
        end_if
end_while
output counter;
```

Suppose we want to estimate the mean of the number of test cases required to reveal the first failure, that is, the F-measure, for FSCS with an accuracy of $\pm r\%$ and a confidence level of $(1-\alpha)\times 100\%$, where $1-\alpha$ is the confidence coefficient. According to the central limit theorem, the size of S required to achieve this goal should be at least as

$$|S| = \left(\frac{100 \cdot z \cdot \sigma^a}{r \cdot \mu^a}\right)^2 \tag{1}$$

where z is the normal variate of the desired confidence level, μ^a is the population mean and σ^a is the population standard deviation. Similarly, for RT,

$$|S| = \left(\frac{100 \cdot z \cdot \sigma^r}{r \cdot \mu^r}\right)^2 \tag{2}$$

where μ^r and σ^r are the population mean and standard deviation, respectively. To ensure the size of S satisfying both FSCS and RT, we take the maximum value of equations (1) and (2), that is

$$|S| = max\left[\left(\frac{100 \cdot z \cdot \sigma^a}{r \cdot \mu^a}\right)^2, \left(\frac{100 \cdot z \cdot \sigma^r}{r \cdot \mu^r}\right)^2\right] \tag{3}$$

Obviously, the larger the sample size $|S|$, the higher the associated confidence z, or the smaller the accuracy range r. However, larger samples mean more effort

and resources. In our experiment, the confidence level was set to 95% and r was set to 5%. In other words, we collect samples until the sample mean is accurate within 5% of its value at 95% confidence. From the statistical tables, we know that for 95% confidence, $z = 1.96$. Moreover, since μ^a, μ^r, σ^a and σ^r are unknown, their estimators $\bar{u}^a$, $\bar{u}^r$, s^a and s^r were used instead, respectively, where $\bar{u}^a = \frac{\sum_{i=1}^{n} u_i^a}{n}$, $\bar{u}^r = \frac{\sum_{i=1}^{n} u_i^r}{n}$, s^a is the standard deviation of $\{u_1^a, u_2^a, \cdots, u_n^a\}$, s^r is the standard deviation of $\{u_1^r, u_2^r, \cdots, u_n^r\}$ and n is the current size of S. Thus, equation (3) becomes

$$|S| = max\left[\left(\frac{100 \cdot 1.96 \cdot s^a}{5 \cdot \bar{u}^a}\right)^2, \left(\frac{100 \cdot 1.96 \cdot s^r}{5 \cdot \bar{u}^r}\right)^2\right] \tag{4}$$

The above equation was used to decide when the process of collecting (u_i^a, u_i^r) could be stopped.

As shown in Table 2, the sizes of S vary with programs, but less than 3000. In this experiment, we have chosen 3000 as the sample size for all programs to calculate the means and standard deviations for both FSCS and RT.

Table 2. Mean Comparison Summary

Pg ID	k	$\|S\|$	F_a	95% CI of F_a	F_r	95% CI of F_r
AIRY	10	1506	799.29	(779.43, 819.15)	1381.44	(1332.51, 1430.37)
BESSJ	10	1598	466.66	(451.95, 481.38)	802.17	(722.90, 831.44)
BESSJ0	10	1567	423.93	(412.55, 435.30)	733.96	(707.44, 760.48)
CEL	10	1550	1607.80	(1552.49, 1663.11)	3065.25	(2955.10, 3175.40)
EL2	10	1650	686.48	(661.78, 711.17)	1430.76	(1377.71, 1483.81)
ERFCC	10	1543	1004.68	(980.34, 1029.02)	1803.62	(1738.94, 1868.30)
GAMMQ	10	1557	1081.43	(1044.35, 1118.51)	1220.28	(1176.32, 1264.24)
GOLDEN	10	1569	1829.74	(1765.68, 1893.80)	1860.81	(1793.52, 1928.10)
PLGNDR	10	1438	1806.94	(1754.47, 1859.41)	2741.66	(2646.75, 2836.57)
PROBKS	10	1480	1442.50	(1407.24, 1477.76)	2634.86	(2542.32, 2727.40)
SNCNDN	10	1617	628.68	(606.16, 651.20)	636.19	(612.83, 659.54)
TANH	10	1469	306.86	(299.21, 314.51)	557.96	(538.43, 577.48)

where

Pg ID	Program ID
k	Size of the candidate set
$\|S\|$	Size of the sample set
F_a	Mean of $\{u_1^a, u_2^a, \cdots, u_{\|S\|}^a\}$
95% CI of F_a	95% Confidence Interval of FSCS mean
F_r	Mean of $\{u_1^r, u_2^r, \cdots, u_{\|S\|}^r\}$
95% CI of F_r	95% Confidence Interval of RT mean

Table 2 lists the sample means within an accuracy of 5% and a confidence level of 95%, for all 12 programs. Also included are their 95% confidence interval (CI).

Table 2 shows that in all 12 programs the F_a's are smaller than the corresponding F_r's, that is, on the average, FSCS required fewer test cases than RT to reveal the first failure. Furthermore, in 10 programs (AIRY, BESSJ, BESSJ0, CEL, EL2, ERFCC, GAMMQ, PLGNDR, PROBKS and TANH), the corresponding FSCS's and RT's 95% confidence intervals do not overlap each other. This implies that F_a is smaller than F_r with a probability of $(0.95)^2$ at least. For the remaining 2 programs (GOLDEN and SNCNDN), the 95% confidence intervals of F_a and F_r overlap each other.

To calculate the performance improvement, we use the following formula:

$$\frac{F_r - F_a}{F_r} \times 100 \tag{5}$$

Table 3 shows that the performance improvement for FSCS over RT ranges from the best end of 52.02% to the worst end of 1.18%. Nine programs (AIRY, BESSJ, BESSJ0, CEL, EL2, ERFCC, PLGNDR, PROBKS and TANH) show a very significant improvement; one program (GAMMQ) shows a moderate improvement; two programs (GOLDEN and SNCNDN) show small improvement.

In summary, FSCS has a considerably smaller F-measure than RT. The experimental results show that FSCS has a good chance at outperforming RT by a very significant margin, which can be as high as 50%. On the other hand, FSCS needs more resources to select the next test case from the candidate set than RT does. Let n be the value of the F-measure and k be the size of the candidate set in FSCS. The selection overheads is of the order kn^2.

5 Conclusion

Random testing is simple in concept and is easy to be implemented. Besides, it can infer reliability and statistical estimates. Hence, it has been used by many testers. Since random testing does not make use of any information to generate test cases, it may not be a powerful testing method and its performance is solely dependent on the magnitude of failure rates.

Table 3. Improvement of the F-measure of FSCS over RT

Pg ID	Improvement in %
AIRY	42.14
BESSJ	41.83
BESSJ0	42.24
CEL	47.55
EL2	52.02
ERFCC	44.30
GAMMQ	11.38
GOLDEN	1.67
PLGNDR	34.09
PROBKS	45.25
SNCNDN	1.18
TANH	45.00

A recent study has shown that failure patterns may be classified as point, strip or block failure patterns. Intuitively speaking, when the failure pattern is not a point pattern, more evenly spread test cases have a better chance of hitting the failure patterns. Based on this intuition, we propose a modified version of random testing called *adaptive random testing*. An empirical analysis of 12 published programs has shown that adaptive random testing outperforms random testing significantly for most of the cases.

Our experimental results have been very encouraging, providing evidences that our intuition of spreading test cases more evenly within the input space is potentially very useful. Nevertheless, there are a number of issues of adaptive random testing that need to be considered, such as various criteria of evenly spreading of test cases, ways of defining the candidate sets. We anticipate that analysis of these issues would further improve the effectiveness of adaptive random testing. In fact, we have already obtained some very interesting results [9, 10].

Acknowledgment

We would like to thank F. T. Chan, T. H. Tse and Z. Q. Zhou for their invaluable discussions. We are also grateful to Dave Towey, who has converted the programs used in the former experiment into C++ languages and rerun the experiment.

T. Y. Chen is particularly indebted to Jean-Louis Lassez for showing how to be a model mentor, how to identify the incompleteness of an apparently complete solution, and how to find simple solutions.

References

1. White, L.J.: Software testing and verification. Advances in Computers **26** (1987) 335–391
2. Hamlet, R.: Random testing. Encyclopedia of Software Engineering. Edited by Marciniak, J. Wiley (1994)
3. Chan, F.T., Chen, T.Y., Mak, I.K., Yu, Y.T.: Proportional sampling strategy: guidelines for software testing practitioners. Information and Software Technology **38** (1996) 775–782
4. Chen, T.Y., Yu, Y.T.: On the relationship between partition and random testing. IEEE Transactions on Software Engineering **20** (1994) 977–980
5. White, L.J., Cohen, E.I.: A domain strategy for computer program testing. IEEE Transactions on Software Engineering **6** (1980) 247–257
6. Association for Computing Machinery: Collected Algorithms from ACM, Vol. I, II, III. Association for Computing Machinery (1980)
7. Press, W.H., Flannery, B.P., Teukolsky, S.A., Vetterling, W.T.: Numerical Recipes. Cambridge University Press (1986)
8. Freund, J.E.: Modern Elementary Statistics. Fifth edn. Prentice–Hall (1979)
9. Chen, T.Y., Kuo, F.C., Merkel, R.G., Ng, S.P.: Mirror adaptive random testing. Information and Software Technology (Accepted for publication)
10. Chen, T.Y., Eddy, G., Merkel, R., Wong, P.K.: Adaptive random testing through dynamic partitioning. In: Proceedings of the 4th International Conference on Quality Software (QSIC 04), IEEE Computer Society Press (2004)