An Innovative Approach to Tackling the Boundary Effect in Adaptive Random Testing *

T. Y. Chen
Swinburne University of Technology
Hawthorn, Australia
tchen@ict.swin.edu.au

De Hao Huang †
Swinburne University of Technology
Hawthorn, Australia
dhuang@ict.swin.edu.au

T. H. Tse
The University of Hong Kong
Pokfulam, Hong Kong
thtse@cs.hku.hk

Zongyuan Yang
East China Normal University
Shanghai, P. R. China
yzyuan@cs.ecnu.edu.cn

Abstract

Adaptive Random Testing (ART) is an effective improvement of Random Testing (RT) in the sense that fewer test cases are needed to detect the first failure. It is based on the observation that failure-causing inputs are normally clustered in one or more contiguous regions in the input domain. Hence, it has been proposed that test case generation should refer to the locations of successful test cases (those that do not reveal failures) to ensure that all test cases are far apart and evenly spread in the input domain. Distance-based ART and Restricted Random Testing are the first two previous attempts. However, test cases generated by these attempts are far apart but not necessarily evenly spread, since more test cases are generated near the boundary of the input domain. This paper analyzes the cause of this phenomenon and proposes an enhanced implementation based on the concept of virtual images of the successful test cases. The results of simulations show that the test cases generated by our enhanced implementation are not only far apart but also evenly spread in the input domain. Furthermore, the fault detection capability of ART for high-dimensional input domains is also enhanced.

1. Introduction

1.1. Random Testing

In a typical commercial software development organization, testing often accounts for over 50% of the total development cost. Since exhaustive testing is infeasible in most situations, research has been focused on the selection of test cases that have higher chances of revealing program failures [13]. Among the test case selection strategies, random testing (RT) is regarded as a simple but useful method [14, 15]. It avoids complex analyses of program specifications or structures and simply selects test cases from the whole input domain randomly. Hence, the test case generation process is cost effective and can be fully automated. RT has been successfully applied in many real-life applications [8, 9, 11, 16–18, 21–23]. For example, it is used as an effective test case generator to test the robustness of Windows NT applications [11], Java JIT compiler [23], database systems [22], and several versions of UNIX system [16, 17]. Furthermore, industry has noticed its importance and begins to incorporate it in software testing tools [1].

1.2. Successful Test Cases

If a test case does not reveal any failure, it is regarded as a successful test case. Most test cases are successful test cases if the program is written by a competent programmer [10]. In conventional testing, successful test cases are usually considered to be useless [20] and will be discarded or retained only for regression testing later.

* This project is partially supported by a Discovery Grant of the Australian Research Council (project no. DP0557246).

† All correspondence should be addressed to: De Hao Huang, Faculty of Information & Communication Technologies, Swinburne University of Technology, Hawthorn 3122, Australia. Email: dhuang@ict.swin.edu.au

Proceedings of the 40th Annual Hawaii International Conference on System Sciences (HICSS'07)
0-7695-2755-8/07 $20.00 © 2007 IEEE
1530-1605/07 $20.00 © 2007 IEEE

However, in our view, successful test cases are informative and should be exploited further. Fault-based testing [19] is an example of the utilization of successful test cases to prove the absence of specific kinds of error.

1.3. Adaptive Random Testing

Recently, Chen et al. proposed a method named Adaptive Random Testing (ART) to improve on the fault detection capability of RT by exploiting successful test cases [6]. ART is based on the observation [7] that failure-causing inputs are clustered together in one or more regions. In other words, failure-causing inputs are "denser" in some areas than others. In general, common failure-causing patterns can be classified into the block, strip, and point patterns. Examples of these failure patterns for a program with a 2-dimensional input domain are given in the schematic diagrams in Figure 1, where the outer square represents the input domain and the shaded areas represent failure-causing inputs. Figure 2(a-c) show fragments of pseudo-code producing each of these types of failure pattern. Intuitively, subsequent test cases that are close to successful test cases are less likely to hit the failure-causing region than those that are far apart from successful test cases. Hence, ART exploits the spatial distribution of successful test cases to ensure that test cases be evenly spread and far apart from one another.

Simulations and empirical studies of real-life programs [6] have shown that ART have significantly enhanced RT in the sense that fewer test cases are needed to detect the first failure. Chen et al. [7] also proposed to use F-measure, the number of test cases to detect the first failure, as the metric for fault detection capability. They reason that F-measure is a more informative metric because testing usually stops after the first failure has been detected.

Several ART algorithms have been proposed [2–4] based on the same rationale. Distance-based ART (DART) and Restricted Random Testing (RRT) are the first two attempts. However, these two algorithms have a general preference in generating test cases close to the edge of the input domain. In other words, the test cases generated by these implementations are not evenly spread. Consequently, the fault detection capability depends on the location of the failure-causing region.

In this paper, we analyze the cause of this phenomenon and propose an enhancement to the DART and RRT algorithms. Section 2 analyzes the root of this phenomenon. Section 3 proposes the enhanced algorithms and Section 4 reports on simulation results. The conclusion is given in Section 5.

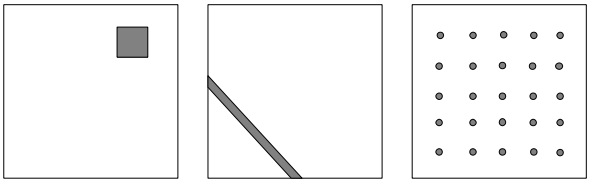

Figure 1. Examples of the three types of failure pattern

```
INTEGER X, Y
INPUT X, Y
IF (X > 7 AND X < 9)
        AND (Y > 8 AND Y < 12)
THEN
        Z = X + Y
        // should be Z = X * Y
ELSE
        Z = X / Y
OUTPUT Z
```

(a) Block pattern

```
INTEGER X, Y
INPUT X, Y
IF (X + Y < 10)
        // should be IF (X + Y < 12)
THEN
        Z = X * Y
ELSE
        Z = X / Y
OUTPUT Z
```

(b) Strip pattern

```
INTEGER X, Y
INPUT X, Y
IF (X mod 4 = 0)
        AND (Y mod 4 = 0)
THEN
        Z = X + Y
        // should be Z = X * Y
ELSE
        Z = X / Y
OUTPUT Z
```

(c) Point pattern

Figure 2. Code fragments producing examples of the three types of failure pattern

Set S to be empty and l to be 0;
do {
  Randomly generate k test cases
  to form the candidate set C;
  for each candidate C_i in C {
    for each successful test case S_j in S {
      $dist(C_i, S_j) = \sqrt{\sum_{p=1}^{n} (c_{i_p} - s_{j_p})^2}$
    }
    $Min_i = \min\{dist(C_i, S_j) | 1 \leq j \leq l\}$
    /*Mini is the minimum Cartesian distance
    between test case candidate C_i and all
    successful test cases in S*/
  }
  Take C_q as the test case such that
    $Min_q = \max\{Min_i | 1 \leq i \leq k\}$
  Add C_q to S
  $l = l + 1$;
} while (C_q does not reveal a failure
  and resource limit has not been reached)

Figure 3. The original DART algorithm

2. Boundary Effects in Some ART Implementations

The rationale behind ART is to achieve an even spread of test cases by exploiting the spatial distribution of successful test cases. Based on this rationale, several implementations of ART have been developed. However, most of them cannot achieve a truly even distribution of test cases. An analysis of these implementations will be presented in this section.

Distance-based ART (DART) [6] is the first implementation of ART. This method maintains a set of candidate test cases $C = \{C_1, C_2, \ldots, C_k\}$ and a set of successful test cases $S = \{S_1, S_2, \ldots, S_l\}$. The candidate set consists of a fixed number of test case candidates, from which new test cases will be selected. The successful set records the locations of all successful test cases, which are used to guide the selection of the next test case. For each test case candidate C_i, DART computes its distance d_i from the successful set (defined as the minimum distance between C_i and the successful test cases), and then selects the candidate C_i having the maximum d_i to be the next test case. The algorithm is shown in Figure 3.

It should be noted that candidates located close to the boundary of the input domain have a higher chance to be selected as test cases than those close to the center, because no successful test cases can be outside the boundary. In this paper, we refer to the phenomenon that the test cases are more likely to be clustered near domain boundaries as the *boundary effect*.

Restricted Random Testing (RRT) [2] is another

Set S to be empty, l to be 0;
do {
  do {
    Randomly generate a test cases candidate c;
    for each successful test case S_j in S {
      $dist(c, S_j) = \sqrt{\sum_{p=1}^{n} (c_p - s_{j_p})^2}$
      if dist (c, S_j) > exclusion zone radius of S_j
        c is outside the exclusion zone
      else
        c is inside the exclusion zone
      }
    } while (c is not outside all the exclusion zone)
  c is the next test case
  Add c to S
  $l = l + 1$;
} while(c does not reveal a failure
  and the resource limit has not been reached)

Figure 4. The original RRT algorithm

implementation of ART. It only maintains the successful set $S = \{S_1, S_2, \ldots, S_l\}$ without any candidate set. Instead, RRT specifies exclusion zones around every successful test case. It randomly generates test case one by one until a candidate outside all exclusion zones is found. The algorithm is shown in Figure 4.

Both DART and RRT select test cases based on the locations of successful test cases, and use distances as a gauge to measure whether the next test case is sufficiently far apart from all successful test cases. Hence, the boundary effect also exists in RRT. The candidates near the boundary have a higher chance to be outside all exclusion zones.

Two series of simulations were conducted to demonstrate such effect. In the first simulation, we investigated the spatial distributions of the test cases generated by DART and RRT without considering the failure-causing inputs. In each trial of test case generation, the locations of the first n test cases, where n = 1, 2, 3, 4, 5, 10, 15, 20, 25, 30, 40, 50, 100, 500, or 1000, were recorded. A million independent trials for DART and RRT were conducted. The spatial distributions were studied for the first n test cases of each trial for the respective values of n.

To clearly demonstrate the spatial distribution, the positions of the test cases were projected onto one dimension. We analyzed the distribution in one dimension without loss of generality because ART algorithms treat every dimension independently. The simulation was conducted in a 2-dimensional input domain in the shape of a unit square. The test case distributions in one dimension are illustrated as histograms with equal bins of size 0.01, consisting of 0 to 0.01, 0.01 to 0.02, and so on. The number of test cases that reside within each bin

Proceedings of the 40th Annual Hawaii International Conference on System Sciences (HICSS'07)
0-7695-2755-8/07 $20.00 © 2007 IEEE

Location of failure-causing input	DART	RRT
Edge	53	50
Center	67	65
Anywhere	66	62

Table 1. Average F-Measure of differently located failure-causing inputs for DART and RRT under block failure pattern on a 2-dimensional input domain (θ = 0.01 on an average of 5000 trials)

is computed. For a fair comparison of the distributions in different test case generation stages, the numbers of test cases in the histograms were normalized to $1/n$ of the actual numbers. Figure 5 illustrates the histograms for DART. The histograms of RRT are not listed, as they are similar to DART. It can be seen that test cases always prefer to be close to the boundary of the input domain, but the preferred region becomes narrower with the increase of test cases.

The second simulation investigates the fault detection capability if the locations of failure-causing inputs were purposely controlled to be close to the boundary or center. The locations of failure-causing inputs are classified as the center area or edge area as follows: The center area ("Center") is defined as the central 80% of the whole input domain and the other area is defined as the edge area ("Edge"). In the simulations, a square failure-causing region with failure rate 0.01 was randomly assigned anywhere in the input domain ("Anywhere") or confined to specified areas (namely "Center" or "Edge"). For both DART and RRT, the simulation was conducted in a 2-dimensional input domain. Table 1 lists the average F-measure of 5000 trials for controlled failure-causing regions. The results indicate that both DART and RRT have higher fault detection capabilities when the failure-causing inputs are close to the boundary.

3. Enhancement of ART Implementations

3.1. Virtual Images of Successful Test Cases

This paper proposes an approach to tackling the boundary effect of ART implementations. As analyzed in the last section, the reason for the boundary effect is that no successful test cases can be outside the input domain. Our approach introduces a new concept of virtual images of successful test cases. Intuitively, the virtual image can be constructed by shifting the input domain. Consider, for example, a 2-dimensional square input domain, as shown in Figures 6(a) to 6(h). The squares with solid lines represent the original input domain with an input range of

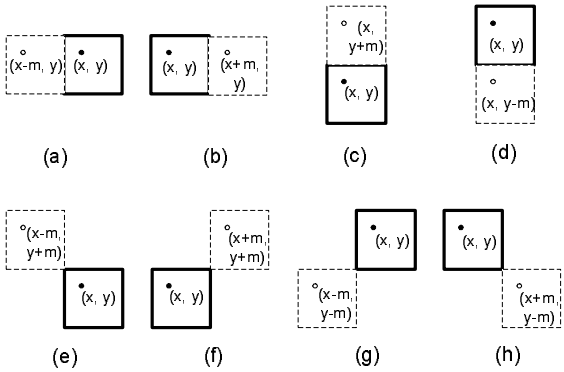

Figure 6. Virtual image construction process in 2-dimensional square input domain

m, and the solid dots represent a successful test case (x,y). For example, Figure 6(a) shows that the input domain is virtually shifted left horizontally by a distance of m. The squares with dashed lines represent the virtual images of the input domain, and the hollow dot represents the virtual images of the successful test case. By a horizontal left shift, a virtual image $(x-m,y)$ is introduced outside the input domain. The 2-dimensional input domain can be shifted along one dimension or both dimensions. Figures 6(a) to 6(d) show shifts along one dimension whereas Figures 6(e) to 6(h) show shifts along both dimensions. There are a total of 9 virtual images of the successful test case (x,y). They are $(x-m,y),(x+m,y),(x,y+m),(x,y-m),(x-m,y+m),(x+m,y+m),(x-m,y-m),(x+m,y-m)$, and (x,y). It should be noted that the original test case can also be regarded as an image of itself.

For an n-dimensional input domain, let $s = (s_1,\ s_2,\ \ldots,\ s_n)$ be a successful test case and $(m_1,\ m_2,\ldots,\ m_n)$ be the ranges of the input domain. Let $v = (v_1,\ v_2,\ \ldots,\ v_n)$ be a virtual image of s that can be computed from its original coordinates and the offset $o = (o_1,\ o_2,\ \ldots,\ o_n)$ as follows:

$$v_i = s_i + o_i$$

where $o_i = -m_i, 0, m_i$ for $i = 1,2,\ldots,n$. Obviously, in an n-dimensional input domain, there are 3^n virtual images of a successful test case.

3.2. Effective Image in Distance Computations

Our enhancement in the ART implementations is based on the concept of virtual images of successful test cases. Whenever the locations of the successful test cases are considered, not only the locations of the originals but also locations of the virtual images are taken into account. In previous implementations, the distance computations

Proceedings of the 40th Annual Hawaii International Conference on System Sciences (HICSS'07)
0-7695-2755-8/07 $20.00 © 2007 IEEE

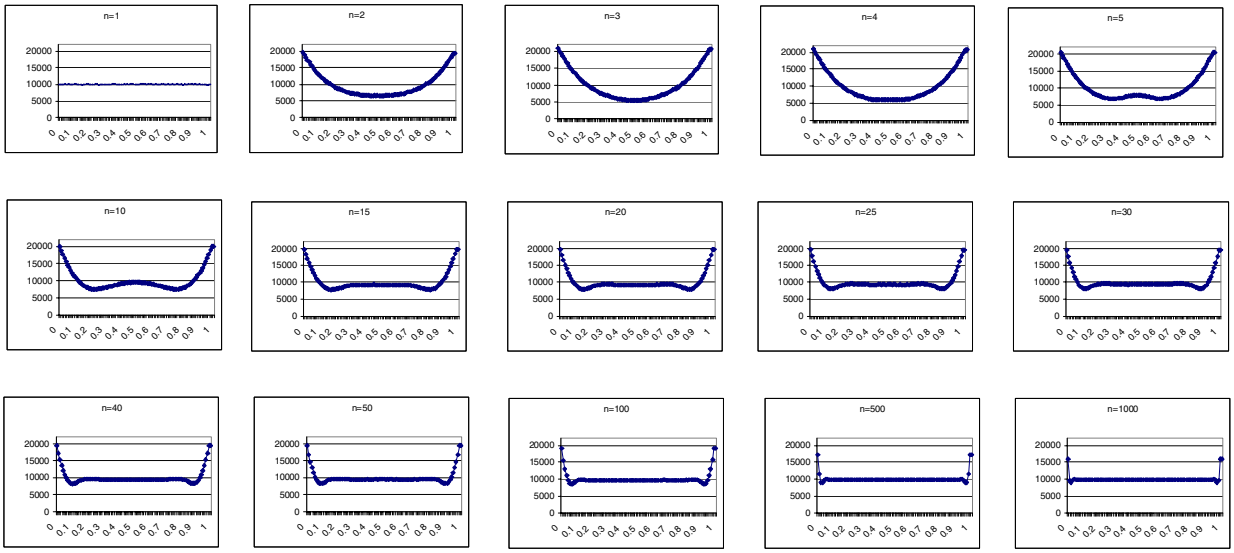

Figure 5. Histogram of DART test cases in one dimension. The *x*-axis represents the locations of test cases. The *y*-axis represents the number of test cases per bin of size 0.01.

only covered actual successful test cases. For example, in an n-dimensional input domain, the distance between a successful test case $s = (s_1,\ s_2,\ \ldots,\ s_n)$ and a candidate $c = (c_1,\ c_2,\ \ldots,\ c_n)$ is calculated as follows:

$$dist(s,c) = \sqrt{\sum_{i=1}^{n}(s_i - c_i)^2}$$

The enhanced implementations use the effective image of the successful test case rather than the successful test case itself in distance computations. The effective image $e = (e_1,\ e_2,\ \ldots,\ e_n)$ of a successful test case s with respect to candidate c is defined as the virtual image of s that has the minimum distance from c. The same successful test case has different effective images for different candidates. It should be noted that the identification of effective images does not require the computation of the distance between every virtual image and the candidate. On the contrary, if a virtual image has the minimum offset to the candidate c in each dimension, then this image will automatically have the minimum distance from c. As mentioned before, $e_i (i = 1, 2, \ldots, n)$ can only have a value of $s_i, s_i + m_i$ or $s_i - m_i$. With respect to candidate c, the minimum offset in the i^{th} dimension is

$$\begin{cases} s_i - c_i & \text{if } |s_i - c_i| \le m_i/2 \\ s_i + m_i - c_i & \text{if } |s_i - c_i| > m_i/2 \text{ and } s_i < c_i \\ s_i - m_i - c_i & \text{if } |s_i - c_i| > m_i/2 \text{ and } s_i > c_i \end{cases}$$

Therefore, we know that the effective image $e = (e_1,\ e_2,\ \ldots,\ e_n)$ has the following property:

$$e_i = \begin{cases} s_i & \text{if } |s_i - c_i| \le m_i/2 \\ s_i + m_i & \text{if } |s_i - c_i| > m_i/2 \text{ and } s_i < c_i \\ s_i - m_i & \text{if } |s_i - c_i| > m_i/2 \text{ and } s_i > c_i \end{cases}$$

Consequently, the distance computation in the enhanced implementations is changed to

$$dist(s,c) = \sqrt{\sum_{i=1}^{n}(s_i - e_i)^2}$$

As an example of illustration, consider a 2-dimensional square input domain (Figure 7). The notions are the same as Figure 6 except that the solid triangles represent candidates. For candidate (1), the effective image is $(x + m, y)$, which is the virtual image closest to it. For candidate (2), the effective image is $(x, y - m)$.

3.3. Enhancement of DART

DART makes use of distance as a gauge to measure whether test cases are far apart from one another and selects the candidate with maximum distance between itself and the successful set as the next test case. However, since no successful test case can be outside the input domain, candidates closer to the boundary are more likely to have a maximum distance from the successful set than candidates closer to the center of the input domain. Our enhanced DART algorithm introduces virtual images of the successful test case and uses effective images in distance

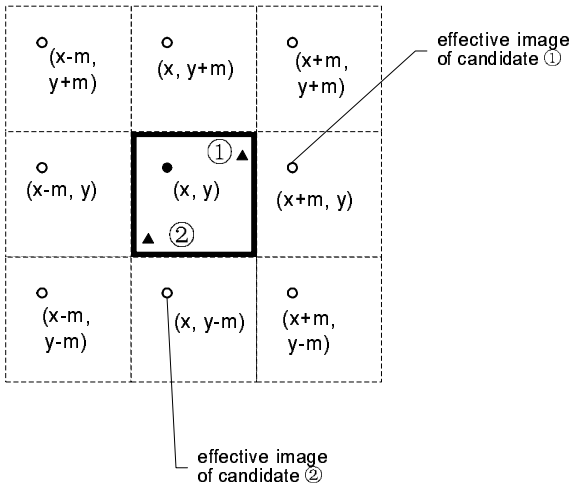

Figure 7. Virtual images of a successful test case in 2-dimensional square input domain

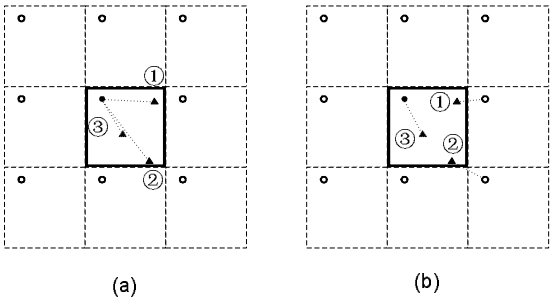

Figure 8. Comparing test case selections between the original and the enhanced versions of DART

computations. Other parts are the same as the previous DART algorithm.

Figures 8(a) and 8(b) compare the original and the enhanced versions of DART in a 2-dimensional input domain. Each of Figures 8(a) and 8(b) puts the original input domain and its 8 images together. The rectangles with solid lines represent the input domain, the solid dots represent the successful test case, and the solid triangles represent the candidates. The rectangles with dashed lines represent the images of the input domain while the hollow circles represent the virtual images of successful test cases. For each candidate in the original DART in Figure 8(a), only the distance from the original successful test case is calculated and, hence, Candidate (2) is selected as the next test case. For each candidate in the enhanced implementation in Figure 8(b), the effective image is identified first. Each dotted line represents the distance between the candidate and its effective image. Suppose candidate (3) has the maximum distance to the effective image of the successful test case comparing with candidates (1) and (2). Then, candidate (3) will be selected as the next test case. As shown in this example, the preference of selecting test cases close to the boundary no longer exists.

3.4. Enhancement of RRT

Although RRT is based on a different intuition, namely that both RRT and DART utilize Euclidean distances to measure how far apart test cases are. Hence, similarly to DART, candidates close to the boundary of the input domain have a higher chance to be outside all exclusion regions than those close to the center.

Similar to the improved DART, RRT can be enhanced to use effective images instead of the original successful test cases in judging whether a candidate is outside the exclusion region. As in DART, among the images of a successful test case, an effective image is defined as the one closest to the candidate. It is obvious that if a candidate is outside the exclusion region of the effective image, it will be outside the exclusion regions of all other images. Hence, it is only necessary to check whether the candidate is outside the exclusion region of the effective image.

Figures 9(a) and 9(b-d) compare the original and enhanced versions of RRT in a 2-dimensional input domain. The notations are the same as those of Figure 8, except that circles with dashed lines are used to denote exclusion regions. In the original RRT shown in Figure 9(a), since candidate (1) is outside the exclusion region, it is selected as the next test case. For each candidate in the enhanced implementation shown in Figure 9(b-d), the effective image is identified first. The images shown as dashed circles are the effective images. If a candidate is outside a dashed circle, it is selected as the next test case. Obviously, candidate (3) will be selected as the next test case. As illustrated in this example, the boundary effect will be reduced, if not totally avoided, by our enhanced version.

4. Simulation Results for the Enhanced DART and RRT Implementations

The main objective of these simulations is to answer the following two questions:

- Are the test cases generated by the enhanced algorithms more evenly spread throughout the input domain?

Proceedings of the 40th Annual Hawaii International Conference on System Sciences (HICSS'07)
0-7695-2755-8/07 $20.00 © 2007 IEEE

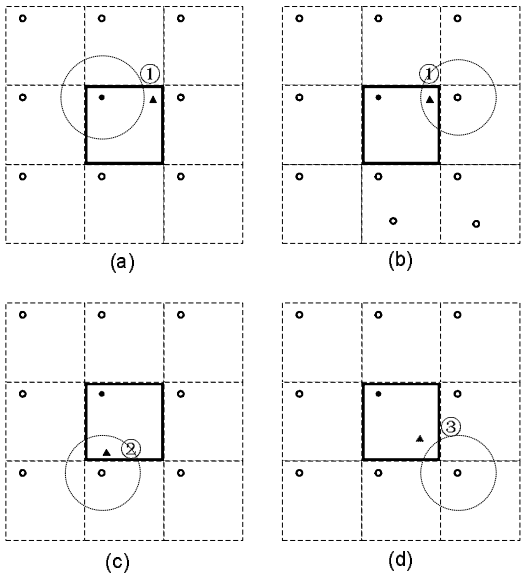

Figure 9. Comparing test case selections between (a) the original and (b–d) the enhanced versions of RRT

- Are the fault detection capabilities of enhanced algorithms better than those of the original algorithms?

To answer the first question, we repeated the distribution analyses in Section 2 for the enhanced DART and RRT. Figure 10 shows the histograms for the enhanced DART. The histograms for the enhanced RRT are again omitted because there is no significant difference from those of the enhanced DART. It is obvious from the figure that the test cases generated by our enhanced algorithms are more evenly spread throughout the input domain in all the test suites under study.

Secondly, we repeated the controlled failure-causing region simulation for the enhanced DART and RRT algorithms. Table 2 lists the average F-measure of 5000 trials. It clearly demonstrates that the fault-detection capabilities for both enhanced versions do not depend on the location of the failure regions.

To compare the fault detection capabilities between the enhanced algorithms and the original ones, simulations were conducted with failure rates 0.01, 0.005, 0.002 and 0.001 for block failure patterns in 2-, 3-, and 4-dimensional input domains. For each combination of failure rate and input domain, 5000 test runs were executed and the average F-measure for each combination was recorded. The fault detection capability of the enhanced DART and RRT outperformed the original versions for every combination of failure rate and input domain. There are two known observations about the original DART and RRT [5]: (a) With the increase of dimensions of the input domain, the fault detection capability decreases dramatically (that is, the F-measure increases). (b) The fault detection capabilities at lower failure rates are better than that at higher failure rates. For the enhanced DART and RRT, the fault detection capability also decreases with the increase of dimensions of the input domain, but the rate is much moderated. Furthermore, the fault detection capability appears to be independent of the failure rates. Obviously, the rectification of the boundary effect has significantly improved on the fault detection capability for DART and RRT.

Location of failure-causing input	DART	RRT
Edge	63	63
Center	62	63
Anywhere	63	63

Table 2. Average F-Measure of differently located failure-causing inputs for DART and RRT under block failure pattern on a 2-dimensional input domain (θ = 0.01 on an average of 5000 trials)

5. Conclusion

Random Testing (RT) is a fundamental testing technique. It simply selects test cases from the whole input domain and, hence, does not incur extensive computational overheads as black-box- or white-box-based test case selection strategies. As reported by practitioners [9, 12, 16–18, 21, 23], RT can effectively detect failures in many real-life applications. Chen et al. observed that failure-causing inputs are often clustered in one or more contiguous regions in the input domain and, therefore, proposed Adaptive Random Testing (ART) to improve on the fault detection capability of RT. Their methods make use of the locations of successful test cases (which do not reveal failures) to enforce an even spread of the subsequent test cases. However, their Distance-based ART (DART) and Restricted Random Testing (RRT) methods have preferences in selecting test cases near the boundary of the input domain (known as the boundary effect). This effect adversely affects the performance of ART, and the impact grows with the increase of dimensions of the input domain.

In this paper, we have analyzed the cause of the boundary effect and proposed an approach to tackle it in DART and RRT. Our approach is based on an innovative concept of virtual images of successful test cases. Simulation results have clearly indicated that the

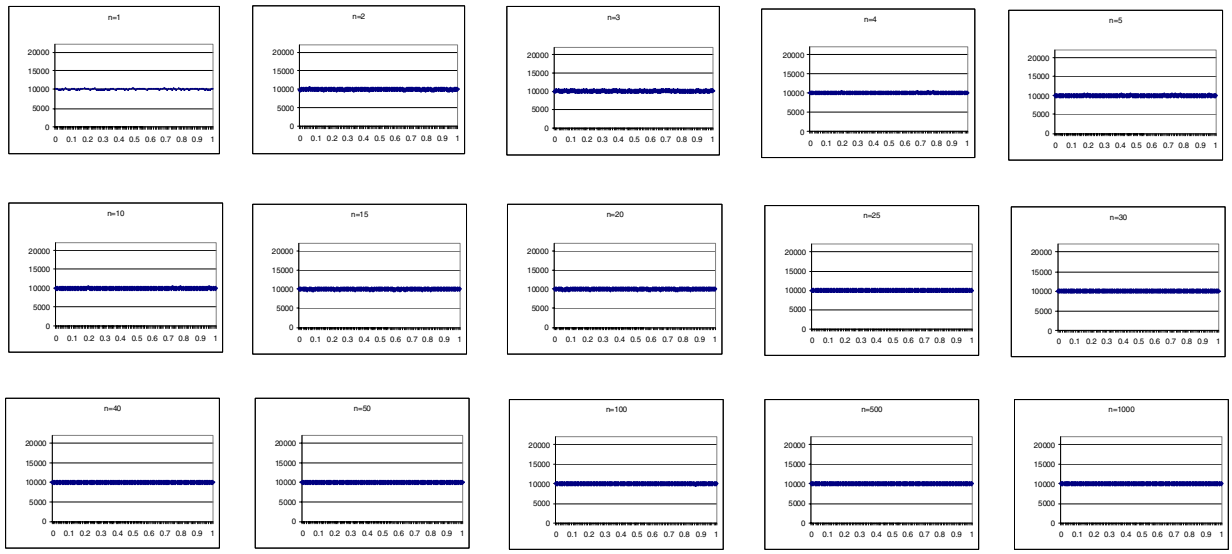

Figure 10. Histograms of the enhanced DART test cases in one dimension. The *x*-axis represents the location of test cases. The *y*-axis represents the number of test cases per bin of size 0.01.

Failure Rate θ	Expected F-measure of RT(F_RT)	2 dimension		3 dimension		4 dimension	
		Mean F_{msr} (F_{ART})	F_{ART}/F_{RT}	Mean F_{msr} (F_{ART})	F_{ART}/F_{RT}	Mean F_{msr} (F_{ART})	F_{ART}/F_{RT}
0.01	100	67	67%	85	85%	108	108%
0.005	200	132	66%	159	80%	196	98%
0.002	500	323	65%	382	77%	475	95%
0.001	1000	648	65%	754	75%	914	91%

Table 3. Average F-Measure of Original ART for block failure pattern (on the average of 5000 trials)

Failure Rate θ	Expected F-measure of RT(F_RT)	2 dimension		3 dimension		4 dimension	
		Mean F_{msr} (F_{ART})	F_{ART}/F_{RT}	Mean F_{msr} (F_{ART})	F_{ART}/F_{RT}	Mean F_{msr} (F_{ART})	F_{ART}/F_{RT}
0.01	100	63	63%	69	69%	75	75%
0.005	200	126	63%	137	69%	150	75%
0.002	500	312	62%	346	69%	371	74%
0.001	1000	632	63%	680	68%	739	74%

Table 4. Average F-Measure of Enhanced ART for block failure pattern (on the average of 5000 trials)

Failure Rate θ	Expected F-measure of RT(F_RT)	2 dimension		3 dimension		4 dimension	
		Mean F_{msr} (F_{ART})	F_{ART}/F_{RT}	Mean F_{msr} (F_{ART})	F_{ART}/F_{RT}	Mean F_{msr} (F_{ART})	F_{ART}/F_{RT}
0.01	100	66	66%	81	81%	95	95%
0.005	200	130	65%	160	80%	185	93%
0.002	500	328	66%	386	77%	453	91%
0.001	1000	644	64%	765	77%	868	87%

Table 5. Average F-Measure of original RRT for block failure pattern (on the average of 5000 trials)

Rate θ / Failure Rate θ	of RT(F_RT) / Expected F-measure of RT(F_RT)	2 dimension		3 dimension		4 dimension	
		Mean F_{msr} (F_{ART})	F_{ART}/F_{RT}	Mean F_{msr} (F_{ART})	F_{ART}/F_{RT}	Mean F_{msr} (F_{ART})	F_{ART}/F_{RT}
0.01	100	63	63%	71	71%	79	79%
0.005	200	126	63%	140	70%	154	77%
0.002	500	319	64%	353	71%	388	78%
0.001	1000	629	63%	706	71%	765	77%

Table 6. Average F-Measure of Enhanced RRT for block failure pattern (on the average of 5000 trials)

test cases generated by our enhanced algorithms are more evenly spread throughout the input domain. As a result, the fault detection capability has also been significantly improved. This improvement is particularly significant for high dimensional input domains.

We plan to apply the concept of virtual images to other ART implementations in future research.

References

[1] D. L. Bird and C. U. Munoz. Automatic generation of random self-checking test cases. *IBM Systems Journal*, 22 (3): 229–245, 1983.

[2] K. P. Chan, T. Y. Chen, and D. P. Towey. Normalized restricted random testing. In *Proceedings of the 8th International Conference on Reliable Software Technologies* (*Ada-Europe 2003*), volume 2655 of Lecture Notes in Computer Science, pages 368–381. Springer, Berlin, 2003.

[3] T. Y. Chen and D. H. Huang. Adaptive random testing by localization. In *Proceedings of the 11th Asia-Pacific Software Engineering Conference* (*APSEC 2004*), pages 292–298. IEEE Computer Society Press, Los Alamitos, California, 2004.

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

[5] T. Y. Chen, F.-C. Kuo, and Z. Q. Zhou. On the relationships between the distribution of failure-causing inputs and effectiveness of adaptive random testing. In *Proceedings of the 17th International Conference on Software Engineering and Knowledge Engineering* (*SEKE 2005*), pages 306–311. ACM Press, New York, 2005.

[6] T. Y. Chen, H. Leung, and I. K. Mak. Adaptive random testing. In *Advances in Computer Science: Higher-Level Decision Making: Proceedings of the 9th Asian Computing Science Conference* (*ASIAN 2004*), volume 3321 of Lecture Notes in Computer Science, pages 320–329. Springer, Berlin, 2004.

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

[8] R. H. Cobb and H. D. Mills. Engineering software under statistical quality control. *IEEE Software*, 7 (6): 45–54, 1990.

[9] T. Dabóczi, I. Kollár, G. Simon, and T. Megyeri. Automatic testing of graphical user interfaces. In *Proceedings of the 20th IEEE Instrumentation and Measurement Technology Conference* (*IMTC 2003*), volume 1, pages 441–445. IEEE Computer Society Press, Los Alamitos, California, 2003.

[10] R. A. DeMillo, R. J. Lipton, and F. G. Sayward. Hints on test data selection: help for the practicing programmer. *IEEE Computer*, 11 (4): 34–41, 1978.

[11] J. E. Forrester and B. P. Miller. An empirical study of the robustness of Windows NT applications using random testing. In *Proceedings of the 4th USENIX Windows Systems Symposium*, pages 59–68. Seattle, Washington, 2000.

[12] P. Godefroid, N. Klarlund, and K. Sen. DART: directed automated random testing. *in Proceedings of the 2005 ACM SIGPLAN Conference on Programming Language Design and Implementation* (*PLDI 2005*), *ACM SIGPLAN Notices*, 40 (6): 213–223, 2005.

[13] B. Hailpern and P. Santhanam. Software debugging, testing, and verification. *IBM Systems Journal*, 41 (1): 4–12, 2002.

[14] R. Hamlet. Random testing. In *Encyclopedia of Software Engineering*, J. J. Marciniak (editor). Wiley, New York, 2002.

[15] P. S. Loo and W. K. Tsai. Random testing revisited. *Information and Software Technology*, 30 (7): 402–417, 1988.

[16] B. P. Miller, L. Fredriksen, and B. So. An empirical study of the reliability of UNIX utilities. *Communications of the ACM*, 33 (12): 32–44, 1990.

[17] B. P. Miller, D. Koski, C. P. Lee, V. Maganty, R. Murthy, A. Natarajan, and J. Steidl. Fuzz revisited: a re-examination of the reliability of UNIX utilities and services. Computer Sciences Technical Report #1268. University of Wisconsin-Madison, Madison, Wisconsin, 1995.

[18] E. Miller. WebSite testing. White Paper. Software Research, Inc., San Francisco, California, 2006. Available at "http://www.soft.com/eValid/Technology/White.Papers/wpaper.testing.pdf".

[19] L. J. Morell. A theory of fault-based testing. *IEEE Transactions on Software Engineering*, 16 (8): 844–857, 1990.

[20] G. J. Myers. *The Art of Software Testing*. Wiley, Hoboken, New Jersey, 2004.

[21] N. Nyman. In defense of monkey testing: random testing can find bugs, even in well engineered software. AutomationJunkies.com, 1999.

[22] D. R. Slutz. Massive stochastic Testing of SQL. In *Proceedings of 24th International Conference on Very Large Data Bases* (*VLDB '98*), pages 618–622. Morgan Kaufmann, San Francisco, California, 1998.

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

Proceedings of the 40th Annual Hawaii International Conference on System Sciences (HICSS'07)
0-7695-2755-8/07 $20.00 © 2007 IEEE