Adaptive Random Testing Based on Two-Point Partitioning

Chengying Mao
School of Software and Communication Engineering,
Jiangxi University of Finance and Economics, 330013 Nanchang, China
Email: maochy@yeah.net

Keywords: adaptive random testing, test data generation, partition, failure pattern, candidate set

Received: October 24, 2011

Test data generation is a key issue in the field of software testing. Adaptive random testing (ART) method has been proposed by Chen et al. to improve the fault-revealing ability of random testing. In the paper, we are mainly concerned with the partitioning-based adaptive random testing and present a new ART based on two-point partitioning. In the new algorithm, the current max-area region is partitioned by the midpoint of two points instead of a single point. The first point is randomly generated, and the second point is picked out from the candidate set according to the farthest distance criterion. In order to compare our algorithm with other two well-known algorithms, the experiments for the case of two-dimension are performed. The results show that our ART-TPP algorithm has a positive improvement for the other two, i.e. ART-RP and ART-BP. Moreover, the appropriate size of candidate set is determined as 2 or 3 based on our sensitivity analysis.

Povzetek: Predstavljena je nova metoda za generiranje podatkov na osnovi dvo-točkovne porazdelitve

1 Introduction

In the past decades, software testing has proved to be an effective way to ensure software quality. As stated by NIST, software errors cost the U.S. economy about $59.5 billion each year, but the testing infrastructure could save 1/3 cost [1]. However, software testing is also a time-consuming and high cost activity in the whole life-cycle of software development. Consequently, it is is necessary to realize the automation of testing activity so as to improve the efficiency. At present, test data generation is recognized as the most difficult for automated software testing.

In fact, test data generation is a search process of selecting the representative data from the input domain of the program under test. In recent years, the most popular way is to use meta-heuristic search (MHS) techniques such as simulated annealing (SA) [2], ant colony optimization (ACO) [3], and generic algorithm (GA) [4], to produce test inputs which can find faults with high probability. But this kind of test data generation method has two limitations: (1) It needs the guideline information about program's internal constructs, and (2) the search process consumes a lot of time due to slow convergence speed. On the other hand, in general, black-box (functional) testing methods, such as random testing and boundary value analysis, can produce test data with high speed and low cost. However, these methods fail to show strong fault revealing capability. Therefore, a possible solution is to rebuild the low-cost functional testing method to generate more effective test inputs.

Random testing (RT) is a naïve method for generating test data, and has been widely adopted by most popular testing tools. However, the size of test data set is very limited in comparison with the whole input space of program under test, so the test inputs generated by RT are not really even distribution yet. In order to overcome this problem, Chen *et al.* proposed an improved method, called *adaptive random testing* (ART) [5, 6], to produce more decentralized test inputs. Their experimental results show that ART can find potential faults faster than the traditional RT. At present, two kinds of ART have been confirmed effective. One is partitioning-based method [6], and the other is distance-based method [7]. In the former method, random partitioning and bisection are two well-known strategies. In the paper, a new partitioning strategy named two-point partitioning is proposed. We believe that it is a useful supplement for the existing ART methods.

The paper is structured as follows. In the next section, it reviews the background of adaptive random testing. It mainly includes two parts: software failure pattern and the basic idea of ART. Then, the two-point partitioning-based ART is addressed in Section 3. In Section 4, some experiments are studied to validate the effectiveness of our method. Finally, the concluding remarks are given in Section 5.

2 Background

2.1 Software failure pattern

In the field of software testing and debugging, the empirical knowledge may play an important role in finding failure-causing input or locating faults. Therefore, it is necessary to summarize the failure pattern [8] or bug pattern [9] so as to generate good fault-revealing test inputs. Generally speaking, the knowledge about location and shape of failure patterns can facilitate black-box testing methods to select test data.

The failure pattern of program under test, in fact, it is the rule of failure-causing inputs of the program. According to Chen and Schneckenburger's analysis [6, 7, 8], the patterns of failure-causing inputs can be classified into three types: block pattern, strip pattern and point pattern. As illustrated in Figure 1, for the block failure pattern, the inputs causing program failure are within a specific area. From the perspective of program code, this kind of fault may lie in the statement block under a compound predicate. For example, if a fault exists in the branch such as `if(a<=x && x<=b && c<=y && y<=d)`, the failure-causing input area can be denoted as $\{(a, b), (c, d)\}$. In the second pattern, i.e. strip failure pattern, the failure may be attributed to predicate fault in a branch. For example, if an expected form of predicate `if(x+y>=k1)` is wrongly written as `if(x+y>=k2)` by a programmer, the failure-causing inputs will lie in a strip, whose width is determined by the value of $|k1 - k2|$. In the last failure pattern, the failure-causing inputs will scatter into some points or small areas in the whole input domain. The corresponding faults may occur in a branch with modulo operation or bitwise operation etc. For instance, if some statements in a branch `if(x%10==0 && y%10==0)` contain faults, the corresponding failure pattern belongs to the point case.

It should be noted that, we only discuss the two-dimensional case in the above analysis. But these three failure patterns are also applicable to other cases, such as one dimension or high dimension.

2.2 Adaptive random testing

As mentioned above, ART attempts to generate test data which can evenly scatter in the input space with the greatest possible. Hence, this method can enhance the fault-revealing capability of test cases. Based on this idea, Chen *et al.* developed a series of ART methods for generating test data set [5, 6, 7]. Furthermore, Ciupa *et al.* have successfully used ART to test object-oriented software, and their ARTOO method [10] can reduce the number of tests generated to reveal the first fault.

All existing ART methods can be classified into two types: distance-based strategy and partitioning-based strategy. In the paper, we only pay attention to the second strategy. Here, we primarily introduce two well-known partitioning algorithms proposed by Chen *et al* [6].

(1) *Random Partitioning Algorithm* (ART-RP). This kind of partitioning algorithm samples test data according to the proportion of region area to whole input space. The basic process of producing test cases can be described as below.

Without loss of generality, here we assume the input domain as a rectangle for two-dimension case. As illustrated in Figure 2(a), the initial test input is randomly selected from the whole input domain. Then, the space is divided into four sub-rectangles according to X and Y coordinates of the initial input point. Next, the max-area region is selected out from them, and the second test input is randomly generated from this area (as shown in Figure 2(b)). At this moment, the whole input domain has been divided into seven regions. Hereafter, the random partitioning is iteratively performed on the current max-area region until the termination condition is satisfied.

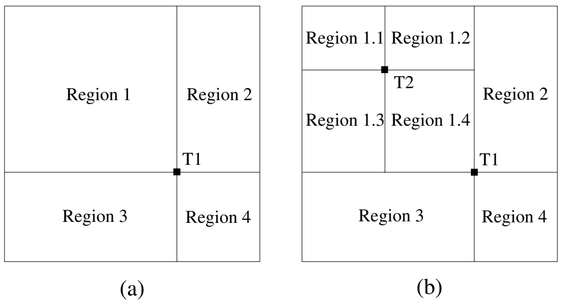

Figure 2: The illustration for random partitioning algorithm (2-*D* case).

(2) *Bisection Algorithm* (ART-BP). The second kind of partitioning strategy proposed by Chen *et al.* is bisection. In this strategy, the partitioning is not based on the coordinates of test input points but the width and height of a region.

Initially, as shown in Figure 3(a), a test input T1 is randomly generated from the whole input domain. Then, the whole region is divided into two parts through performing a partition on the bisector of region height. Meanwhile, another test input T2 is randomly generated in the sub-region, which previously does not contain any test input points (refer to Figure 3(b)). Similarly, the region can also be divided on the bisector of width. As shown in Figure 3(c), test input T3 and T4 can be generated in the next step. Subsequently, the partition process can be continued by alternately bisecting the height and width of each sub-region until the termination condition is satisfied.

2.3 Basic terms

In order to facilitate the expression below, we also follow and define some basic terms about ART. For an input domain D, the corresponding domain size is denoted as d. Meanwhile, we use m and n to denote the number of failure-causing inputs and number of test inputs, respectively. Then, the sampling rate σ and failure rate θ can be defined as n/d and m/d, respectively.

It should be noted that, the case of two-dimension input domain is utilized to describe our partitioning algorithm. For the two-dimension case, a region can be expressed via point P_{ll} and P_{ur}, where P_{ll} represents the lower-left point of the region and P_{ur} is the upper-right point. For each point P, it can be denoted by X and Y coordinates, i.e.

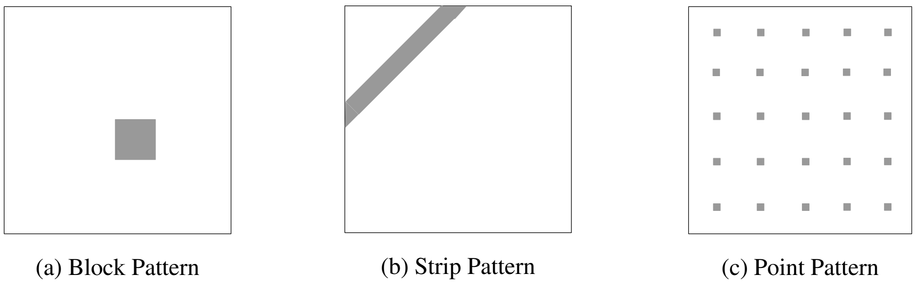

Figure 1: Block, strip, and point failure patterns in a two-dimensional input domain. Here, the shaded areas represent failure-causing inputs.

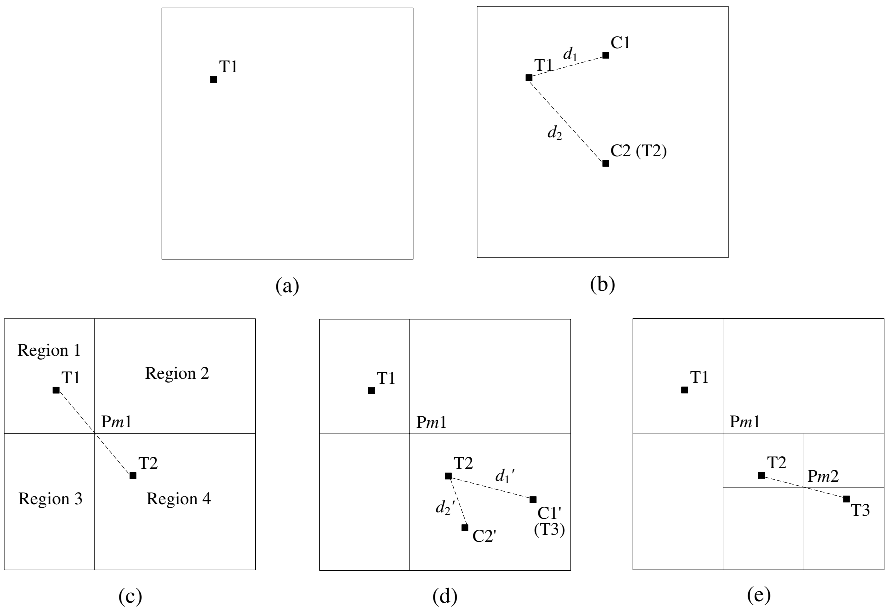

Figure 4: The illustration for the algorithm of two-point partitioning strategy.

P=(x, y). During the process for generating test inputs, the set of test data can be denoted as S_T=$\{T_i|1 \leq i \leq n\}$, and the candidate set of random points is denoted as S_C=$\{C_r|1 \leq r \leq k\}$, where k is the pre-specified size of test input candidate set.

Here, we also use the number of test cases required to detect the first failure (referred to as the F-measure) as the effectiveness metric. For a test data set S_T=$\{T1, T2, \cdots, Tn\}$, if the corresponding F-measure is equal to u $(1 \leq u \leq n)$, which means that the test input between 1 to $u-1$ can't reveal faults but the u-th can find them. Formally, it can be denoted as F=u. With regarding to the traditional random testing, in theory, its F-measure is equal to d/m. When comparing two test data generation methods, the lower value of F-measure means the higher effectiveness, because the low F-measure means few test inputs which are required to reveal the first fault.

3 Two-point Partitioning Algorithm

In the traditional partitioning strategies, it is possible that the two sampled test inputs are very close with each other. Here, we propose a new test data generation strategy based on two points partitioning. The basic process is illustrated in Figure 4, and the steps can be stated as below.

(1) Add the whole input domain into the region list L, and set S_T=Ø.

(2) Select the max-area region from L, denote it as *curReg*, and remove it from L.

(3) If there are no previous test inputs in *curReg*, a new input point should be randomly generated in this region, then add it into S_T. Otherwise, go to step (4).

(4) Suppose the existing test input in *curReg* denoted as T_i, randomly generate k candidate points in *curReg*. Calculate the distance from T_i to each candidate

Algorithm ART-TPP

Input: The boundary point of input domain *bndP*[2], where *bndP*[0] represents the lower-left point of region and *bndP*[1] is the upper-right point.

Output: The set of test data S_T=$\{T1, T2, \cdots, Tn\}$.

```
Stage 1: Initialization
1: regionList = Ø; //regionList is a list of regions
2: set the region (bndP[0], bndP[1]) as curReg; //curReg represents the current region needed to be partitioned
3: tempP = generateRandPoint(curReg.ll, curReg.ur);
4: $S_T$ = $S_T \bigcup \{tempP\}$;
5: add curReg into regionList;
Stage 2: Test Data Generation
6: while true do
7:   pIndex = findMaxRegion(regionList); //find the max-area region in regionList, and pIndex is the index of region needed to be partitioned
8:   curReg = regionList.get(pIndex);
9:   regionList.remove(pIndex); //remove the max-area region from regionList
10:  if curReg doesn't contain an existing test input then
11:    generate a new test input using function generateRandPoint(curReg.ll, curReg.ur), denote it as P1;
12:    $S_T$ = $S_T \bigcup$ {P1};
13:    if P1 hits the failure-causing region then
14:      break;
15:    end if
16:    randomly generate $k$ candidate points in curReg, and store them in $S_C$; //$S_C$ is the test input candidate set
17:    select the point from $S_C$ which is the farthest from P1 as the second test input P2;
18:  else
19:    P1 = the existing test input in curReg;
20:    randomly generate $k$ candidate points in curReg, and store them in $S_C$;
21:    select the point from $S_C$ which is the farthest from P1 as the second test input P2;
22:  end if
23:  $S_T$ = $S_T \bigcup$ {P2};
24:  if P2 hits the failure-causing region then
25:    break;
26:  end if
27:  calculate the midpoint of P1 and P2, divide curReg into four new sub-regions via this midpoint, and then add them into regionList;
28:  locate P1 and P2 to their corresponding sub-regions;
29: end while
30: return $S_T$;
```

point, and select the point which is farthest from T_i as the second test input (denoted as T_{i+1}) in *curReg*, S_T=$S_T \bigcup \{T_{i+1}\}$.

(5) In the step (3) and (4), once a new test input is generated and added into S_T, we should validate whether it can hit the failure-causing region. If true, terminate the process and output S_T. Otherwise, continue the process to append other test inputs.

(6) Compute the midpoint of the corresponding points of T_i and T_{i+1}, denoted as P_{mi}. Then, partition the current max-area region into four sub-regions via P_{mi}, add these sub-regions into L, go to step (2).

Formally, the adaptive random test data generation algorithm based on two-point partitioning can be described in the form of pseudo-code (cf. algorithm **ART-TPP**). It should be noted that, we are mainly concerned with the case of two dimension here. Accordingly, the high dimension case can be treated in the similar way. In the line 3 and 11 of algorithm pseudo-code, function `generateRandPoint`(*curReg*.ll, *curReg*.ur) can randomly generate a test input (or point) in the current region, where *curReg*.ll refers to the lower-left point of region, and *curReg*.ur is the upper-right point. *curReg* is a rectangle region object which contains two main member variables: lower-left point (ll) and upper-right point (ur). The function `findMaxRegion`(*regionList*) in line 7 returns the index of the max-area region in *regionList*. There are several kinds of operations can be invoked by object *regionList*, such as `get`(), `remove`() etc.

Here, we suppose the size of test data set is n. Obviously, the time complexity of function `findMaxRegion`() is $O(l)$, where l is the length of region list, and it varies from 0 to n. Therefore, the time complexity of whole algorithm is $O(n^2/2)$. In general, n is in the same magnitude of θ^{-1} and $n < \theta^{-1}$. Accordingly, the complexity can be expressed as $O(\theta^{-2}/2)$.

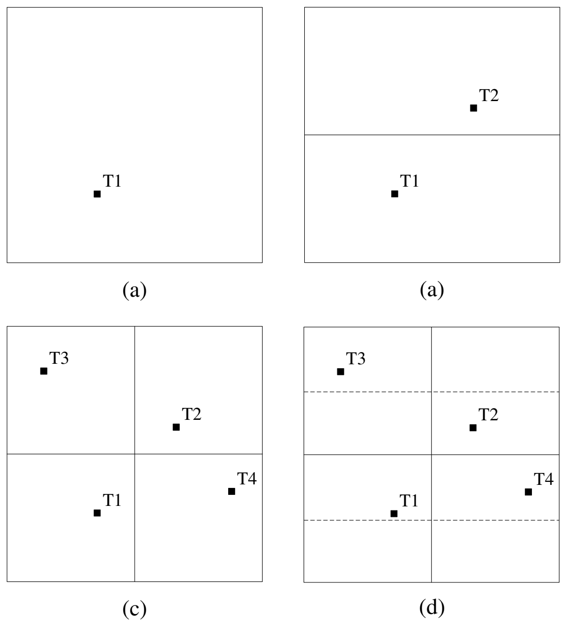

Figure 3: The illustration for bisection algorithm (2-*D* case).

4 Experimental Analysis

4.1 Analysis on failure patterns

In this section, we perform the simulation analysis on the case of two-dimension. The experiment is employed in the environment of Eclipse 3.6 and JRE 1.6.0_05. The program runs on a computer with Pentium IV 1.8 GHz CPU, 1 GB RAM and Windows XP SP2. In this sub-section, we want to investigate the following two questions.

RQ1: How effective is the ART-TPP algorithm for three types of failure patterns?

RQ2: Does the failure rate affect the F-measure ratio (F-ratio for short)?

F-measure ratio = $\frac{F_{ART}}{F_{RT}}$, where F_{ART} represents the F-measure value of ART method, and F_{RT} is the F-measure of the general RT method. According to the description in section 2.3, F_{RT} is equal to $1/\theta$.

As shown in Figure 5, we firstly analyze the F-measure ratios for 20 different random failure regions. For each region, we repeat 5000 runs to get the average value of F-ratio. In the figure, the square-marked curve represents the theoretical value of RT's F-ratio, the other three curves are ART's F-ratios for three kinds of failure patterns. It is not hard to find that, the location of failure region does not impose a great impact on F-ratio. The F-ratio of ART-TPP for block failure pattern is about 76.5%, but its F-ratios of strip and point failure patterns are very close to the theoretical value of RT. This phenomenon is in accordance with Chen's research results [6]. It means that the ART is more effective than the general random testing method, and especially for the block failure pattern.

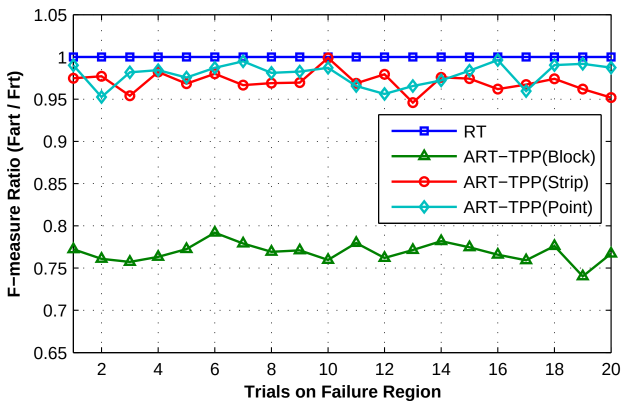

Figure 5: The F-measure ratios for different failure region locations.

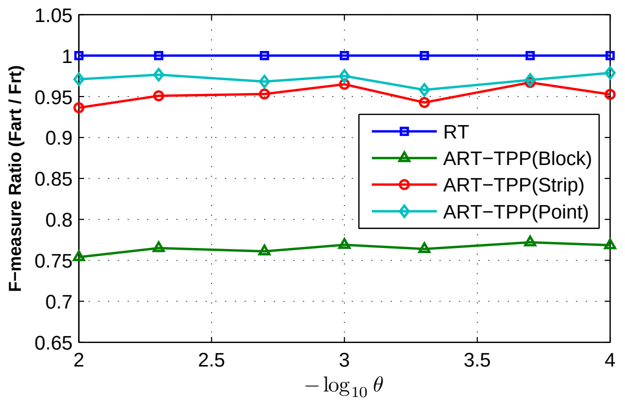

Figure 6: The F-measure ratio vs. failure rate θ.

Meanwhile, we also analyze the F-ratio through varying the failure rate θ from 0.01 to 0.0001 (see Figure 6). For the reason of computing time, when θ is equal to 0.01, 0.005, 0.002 and 0.001, we randomly select 100 locations of failure regions, and repeat 5000 runs for each location to get the average F-ratio. When θ is equal to 0.0005 and 0.0002, 20 random locations and 1000 runs are used for experiments. Moreover, we set 20 random locations and 200 runs when θ is 0.0001. As illustrated in Figure 6, the failure rate does not cause obvious fluctuation on F-ratio. When θ varies from 0.01 to 0.0001, F-ratio approximately keeps the value of 0.765 for block failure pattern, 0.95 for strip pattern and 0.97 for point pattern.

4.2 Comparison analysis

In this sub-section, we want to perform a comparison analysis and answer the question as below.

RQ3: Is our algorithm more effective than the existing two partition-based ART algorithms?

Failure rate	Block pattern			Strip pattern			Point pattern		
	ART-RP	ART-BP	ART-TPP	ART-RP	ART-BP	ART-TPP	ART-RP	ART-BP	ART-TPP
0.01	77.4%	72.1%	75.3%	92.8%	91.9%	93.6%	99.5%	97.7%	97.1%
0.005	77.4%	74.3%	76.5%	97.1%	95.5%	95.0%	99.6%	98.1%	97.9%
0.002	78.0%	73.6%	76.0%	95.3%	95.1%	95.3%	99.7%	98.7%	96.8%
0.001	79.6%	74.1%	76.9%	96.5%	96.6%	96.5%	98.0%	96.8%	97.5%

Table 1: Comparison analysis between ART-TPP and other two ART algorithms

In the experiment, our algorithm is run for different failure rates and different failure region types, the results are shown in Table 1. In the same way, we randomly select 100 locations of failure regions, and repeat 5000 runs for each location to get the average F-ratio. The results of other two algorithms are referred from [6]. Here, we compare the F-ratios of three algorithms as follows.

For the block failure pattern, the effect of ART-TPP algorithm is better than ART-RP but worse than ART-BP for all the values of θ. On average, the F-ratio of our ART-TPP algorithm is lower than that of ART-RP about 2 percent, but greater than that of ART-BP about 2.6 percent.

For the strip failure pattern, the results of ART-TPP are equal to those of other two algorithms on the whole. When θ=0.01, the performance of ART-BP is the best, while ART-TPP is on the worst performance. When θ=0.005, the F-ratio of ART-TPP is the lowest one, and ART-RP acts the worst performance. For the rest values of failure rate (θ), the difference of three algorithms is basically within 0.2 percentage points. The results indicate that, for the strip failure pattern, ART-TPP is quite good in the case of low fault density, but the performance is not good in the case of high fault density.

For the point failure pattern, the F-ratio of ART-TPP is lower and more stable than other two ART methods. When θ is from 0.01 to 0.002, the F-ratio of ART-TPP is always the lowest one in the three algorithms. When θ is equal to 0.001, ART-TPP's F-ratio is higher than that of ART-BP, but lower than that of ART-RP. It is worth noting that the fluctuations of the F-ratios of ART-TPP is the least of three algorithms

Based on the above analysis, we can argue that our ART-TPP algorithm has a positive improvement over the existing two well-known ART algorithms, especially for the random partitioning algorithm (ART-RP).

4.3 Sensitivity analysis

In the above experiments, we fix the candidate set size to 2. In fact, k acts as a parameter of our algorithm, so we have to answer the following question.

RQ4: Does k play an important role in algorithm effectiveness? And which k value is the appropriate choice?

In order to answer this question, we take the block failure pattern as an example to analyze the impact of candidate set size. The results of θ=0.001 and θ=0.0005 are shown in Figure 7(a) and 7(b), respectively. We can find that, the F-ratio value greatly decreases when k is from 1 to 2, i.e. from 0.84 to 0.767. The F-ratio value decreases 0.017 when k is from 2 to 3. When k is greater than 3, the F-ratio has no obvious change and keeps the value on 0.745. Therefore, k=3 is the best value for the size of candidate set, and k=2 is also a suitable choice while considering the computing cost. More importantly, the F-ratio will sharply increase if we choose the second point without candidate selection (i.e. k=1).

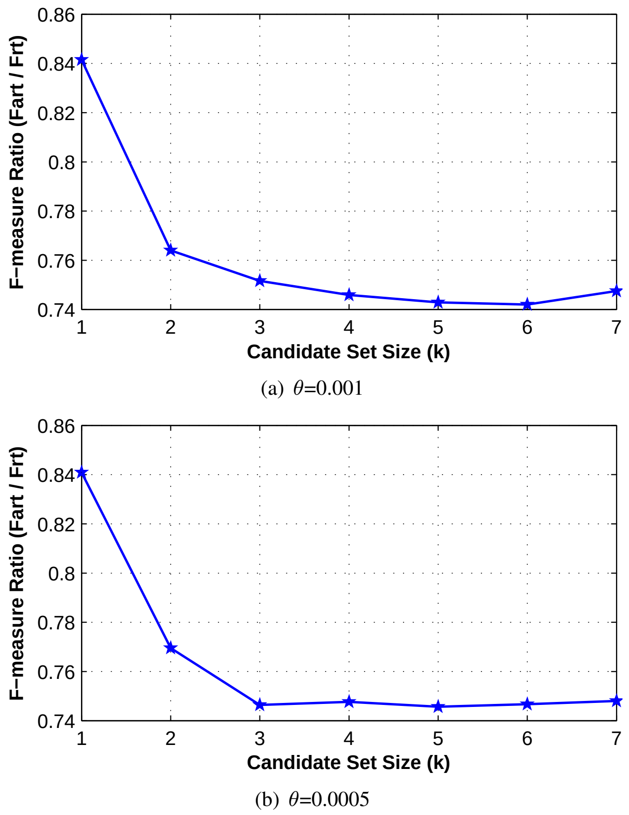

Figure 7: The F-measure ratio vs. candidate set size k.

5 Conclusion

How to generate the test data with high fault-revealing capability is a critical problem in the field of software testing. Random testing has been widely adopted in automated testing tools due to its advantages such as simpleness, easy realization and low cost. Unfortunately, this method usually reveals the potential faults with the large amount of test inputs, so its cost-benefit is not very good. Chen *et al.* proposed an improved strategy named *adaptive random testing* to overcome this shortage. In the paper, we are mainly concerned with the partitioning-based ART. A new algorithm based on two-point partitioning (i.e. ART-TPP) is presented here. In our algorithm, we also select the current max-area region as partition object, but the region is partitioned at the midpoint of two points. At first, we randomly generate a test point in the region. Then, the second point is picked out from a candidate set according to the farthest distance criterion. The partition can be iteratively performed until the potential faults are found or the size of test data set reaches the pre-set limit. In order to validate the effectiveness of ART-TPP algorithm, we compare it with the other two well-known algorithms: random partitioning (ART-RP) and bisection partitioning (ART-BP). The experimental results show that ART-TPP is better than ART-RP but worse than ART-BP in the case of block failure pattern. For the strip failure pattern, the three algorithms have no obvious difference. While considering the point failure pattern, ART-TPP algorithm is better and more stable than other two algorithms.

Of course, there are still some open research issues that need to explored in the next step. For example, we can continue to conduct comparative analysis of the three algorithms in high-dimensional case. Meanwhile, we are planning to use some real-world programs to analyze the effect of our ART-TPP algorithm.

Acknowledgement

This work was supported in part by the National Natural Science Foundation of China (NSFC) under Grant No. 60803046 and 61063013, the Natural Science Foundation of Jiangxi Province under Grant No. 2010GZS0044, the Science Foundation of Jiangxi Educational Committee under Grant No. GJJ10433, the Open Foundation of State Key Laboratory of Software Engineering under Grant No. SKLSE2010-08-23, and the Program for Outstanding Young Academic Talent in Jiangxi University of Finance and Economics.

References

[1] National Institute of Standards and Technology, (2002). *The Economic Impacts of Inadequate Infrastructure for Software Testing*, Planning Report 02-3.

[2] N. Tracey, J. Clark, K. Mander, and J. McDermid, (1998). An Automated Framework for Structural Test-Data Generation, *Proc. of the 13th Int'l Conference on Automated Software Engineering (ASE'98)*, IEEE CS Press, Honolulu, Hawaii, USA, pp. 285–288.

[3] K. Ayari, S. Bouktif, and G. Antoniol, (2007). Automatic Mutation Test Input Data Generation via Ant Colony, *Proc. of the 9th Annual Conference on Genetic and Evolutionary Computation (GECCO'07)*, ACM Press, London, England, UK, pp. 1074–1081.

[4] R. P. Pargas, M. J. Harrold, and R. Peck, (1999). Test-Data Generation Using Genetic Algorithms, *Software Testing, Verification and Reliability*, vol. 9, no. 4, pp. 263–282.

[5] T. Y. Chen, F.-C. Kuo, R. G. Merkel, and T. H. Tse, (2010). Adaptive Random Testing: The ART of Test Case Diversity, *The Journal of Systems and Software*, vol. 83, pp. 60–66.

[6] T. Y. Chen, R. G. Merkel, G. Eddy, and P. K. Wong, (2004). Adaptive Random Testing Through Dynamic Partitioning, *Proc. of the 4th Int'l Conference on Quality Software (QSIC'04)*, IEEE CS Press, Braunschweig, Germany, pp. 79–86.

[7] T. Y. Chen, F.-C. Kuo, and H. Liu, (2007). Distribution Metric Driven Adaptive Random Testing, *Proc. of the 7th Int'l Conference on Quality Software (QSIC'07)*, IEEE CS Press, Portland, Oregon, USA, pp. 274–279.

[8] C. Schneckenburger and J. Mayer, (2007). Towards the Determination of Typical Failure Patterns, *Proc. of the 4th Int'l Workshop on Software Quality Assurance in conjunction with ESEC/FSE'07*, ACM Press, Dubrovnik, Croatia, pp. 90–93.

[9] E. Allen, (2002). *Bug Patterns in Java (2nd Edition)*, Apress.

[10] I. Ciupa, A. Leitner, M. Oriol, and B. Meyer, (2008). ARTOO: Adaptive Random Testing for Object-Oriented Software, *Proc. of the 30th Int'l Conference on Software Engineering (ICSE'08)*, ACM Press, Leipzig, Germany, pp. 71–80.