Fragment Analysis and Test Case Generation using F-Measure for Adaptive Random Testing and Partitioned Block based Adaptive Random Testing

D. Indhumathi
Research Scholar
Department of Information Technology
Bharathiar University
Coimbatore, India

S. Sarala, Ph. D
Assistant Professor
Department of Information Technology
Bharathiar University
Coimbatore, India

ABSTRACT

Test case generation is a path to identify the solution in software testing. Adaptive random testing is an enhancement of random testing to improve the quality of fault-revealing. The research focuses on software adaptive random testing based on Matrix called Partitioned Block based Adaptive Random Testing. It compares the performance of PBART with the existing Adaptive random testing using random samples of test cases which are drawn from blocks of distinct partitions. Partition testing defines as a block of test cases partitioned into set of all test cases. Thereby it has prompted to investigate the performance of random testing that can be improved by taking the patterns of failure-causing inputs which utilizes the prior knowledge and the information of the test cases. The proposed algorithm PB –ART performs the testing of program structure and load the source code to matrix with scenarios, method flows and data values. In numerical experiments, the approach examines effectiveness of PB-ART with ordinary adaptive random testing. There exist three measures for evaluating the effectiveness of a testing technique namely P-measure, E-measure and F-measure. Moreover F-measure is intuitively more appealing to testers and more realistic and informative from a practical point of view. Therefore, F-measure is chosen for measuring testing techniques in this research work.

KEYWORDS

Adaptive random testing, Partition testing, Test case generation, failure pattern, fault detection.

1. INTRODUCTION

Software testing proves the various effective ways to ensure the software quality. Consequently it is necessary to realize the automation of testing activity to improve the efficiency. Software testing is the process of finding errors [1]. Instead testing generates test suites to maximize the probability of fault detection [10].Test case generation is a process of selecting the data from input domain of the program. A successful test case might reveal the presence of failure [2].

The main merits of random testing [3] include the accessibility of efficient algorithms to generate test cases and infer the reliability with statistical measures. In all random testing, the rate of failure-causing inputs is used in the measurement of effectiveness. The test cases may be randomly chosen by uniform distribution or according to the operational problem [14].

Random Testing does not use information about the program under test [8, 9]. Therefore, Adaptive Random Testing has been proposed for common failure patterns in terms of test cases to detect the failure. However, in recent study [4] it has been found that the performance of a partition testing strategy depends not only on the failure rate, it also on the geometric pattern of the failure-causing inputs. The new type of random testing is developed as adaptive random testing which shows that the effectiveness of random testing also it can be improved without incurring significant overheads rather than ordinary random testing.

The test effort refers to complete set of testing is required for software development. The initial process carried out for writing test cases. It specifies functional specification, where the test manager creates a test plan. The test challenge can be divided into three categories namely test case generation, test execution and test evaluation.

The chances of hitting failure patterns depend solely on the magnitude of the failure rate in random testing [7]. In order to inspect the non-point patterns both the strip and block patterns, the failure detection capability can be improved by minimum modification of the ordinary random testing technique using the proposed Partitioned Block ART.

2. RELATED WORK

New Adaptive Random Testing [3, 13] which is an alternative method for random testing to improve the failure detection through failure patterns. ART is based on empirical analysis which shows many program faults in failure contiguous areas. Towards the failure pattern identification by failure based testing results in 50% of the performance improvement.

Adaptive Random testing through Dynamic Partitioning [11] is also to reduce the fixed cost of computations. DP-ART is inspired by partitioning testing, which incrementally divides the input domain to identify the sparsely populated partitions to serve as test case generation region, the two partitioning schemes, namely ART by Random Partitioning and ART by Bisection (B-ART).

The approach [5] tries to construct test data such that a selected criterion, all-nodes-s criterion, gets satisfied. In contrast to that, particular values of fields in parts of the data structure that processes by corresponding tasks in a task tree cannot influence the order of execution. The order of execution of individual tasks is solely restricted by the task tree structure. Another difference between Structural testing criteria for message-passing parallel programs and the definition of paths in matrix does not have a tree structure. It defines two kinds of paths, namely intra-process and inter-process paths, whereas the proposed work defines a path in a matrix state with element positions.

Under proportional allocation of tests to blocks, partition testing will always perform at least a simple random testing, in terms of variances of estimators and failure detection probability [12]. Moreover, partition testing may outperform adaptive random testing. Path-oriented test case generation is a simplest testing technique which performs the testing at path level [6]. The main goal of using PRT is to apply the principle of uniform selection, to the collection of test data that all trigger the same oath. The main challenge of PRT lies in its ability to build efficiently such a test suite in order to minimize the number of rejects.

An effective test case generation using Anti Random Testing based on measurable distance technique for generating test cases which improves the fault detection capabilities [15]. It can be applied to all type of programs and it employs the location of previously executed test cases. The method is used to focus and apply on a program that has numerical input values.

3. METHODOLOGY

The proposed algorithm PB –ART performs the testing from the .NET Solution manifest file as shown in Fig1. Initially the .net solution file is loaded then the manifest file is read to get the internal program structure. With the program structure read the source code then test cases are generated.

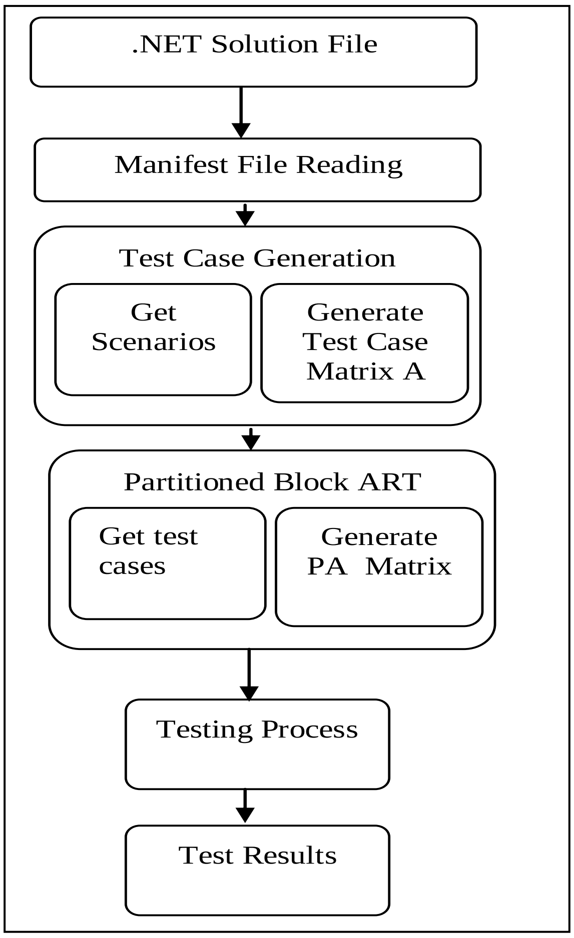

Figure 1. Segmental flow of activities of PB-ART

Test case generation is done by reading the manifest files program structure and then by finding the method flows, callbacks, data values which are used. Test cases are generated with the findings and it is stored in the A matrix. Then PB-ART process is carried out with partitioning the test cases to **n X m** matrix along with tagging of the test execution states.

4. IMPLEMENTATION

For performing the proposed PB-ART, the following steps are included.

1. Loading the .NET Solution Manifest file
2. For each manifest file, generate a full set of test case scenarios.
3. For each scenario, recognize at least one test case and the conditions that will make it execute.
4. For each test case, identify the data values with which to test.
5. Execute the test cases using PB-ART

Figure 2. Implementation of PB-ART

4.1 Loading The .Net Solution File

Test cases choose the collection of code mostly from texts related c# programs. Such texts not only tend to provide a large body of code in a single location, but can also be expected to use a wide range of the language features in the process of explanation. It contrast with sample applications, which provide attention to a subset of the language features, either the concentration on a particular domain of application or because of the coding method of various authors. Manifest file of the c# solution file is read and the method can be parsed through the manifest information.

4.2 Generating Test Cases

A test case is a set of possible inputs, conditions, expected results are developed for specific objective to exercise the program path and verify compliance with the respective requirements. The purpose of generating test cases is to identify and communicate conditions to implement in the test. Test cases are necessary to verify successful and acceptable implementation of the product requirements. It describes the following four-step process for generating test cases from detailed manifest file of the .NET Solution.

```
Initialize matrix A, M, d=0
load the manifest file into M
for each method description d in  M
GenTest(); // call the gentlest method
Output test cases returned into Matrix A
End for
GenTest()
get the test method
iftc>0
identify main and alternate flows of the function
get the scenarios
find data values
mark the test case
return the test case
end if
```

Figure 3. Algorithm1 for generating Test Cases

Step 1: Generate fragment Matrix
Read the manifest file as textual description and identify each combination of main and alternate flows, the scenarios which follows the creation of a fragment matrix. The Table 1 indicates the partial fragment matrix for the input file with .Net Solution. This is a simple example to explain the PB-ART.

TABLE 1: Partial fragment Matrix for the Loaded .NET Solution

Scenario 1 –Database insertion Basic Flow
Scenario 2 –Session Creation Basic Flow
Scenario 3 –Session Ending Basic Flow
Scenario 4 –Bulk Database fetch Basic Flow
Scenario 5 – Modify User Interface Basic Flow
Scenario 6 – Connection Check Basic Flow

Step 2: Identify Test Cases
The possible sets of scenarios have been identified, followed by test cases are generated by analyzing the scenarios and reviewing the use cases of textual description. At least one test case is mandatory for each scenario, but there will probably be more. The textual description for an alternate flow is written as description like,

Connection timeout, session failure and fetch dataset
Thereby the additional test cases may be required to test all the possibilities. In addition add test cases to test boundary conditions. The test cases are to re-read the manifest file textual description and find as the conditions with data elements to execute the various scenarios.

To create a document for the test cases, a matrix format is iterated as the one in Table 2. The top column of the first column contains the test case ID, the second row has a brief description of the test case, including the scenario being tested and all other rows except the last one contain data elements that will be used in implementing the tests. The last row contains a description of expected output.

In this matrix no data values have actually been entered. The cells of the table contain a V, I, or N/A. V indicates valid, I is for invalid, and N/A means that it is not necessary to supply a data value in this case. This specific matrix is a first-rate intermediate step it clearly shows what conditions are being tested for each test case. It is also very easy to determine by looking at the Vs and Is whether it identify a sufficient number of test cases. In addition to the executed scenarios in which everything works fine, each row in the matrix should have at least one I indicating an invalid condition being tested in the test case matrix.

Table 2: Test Cases with Results

Test Case ID	**T1**	**T2**	**T3**	**T4**
Condition	Button Click	Session Creation	Fetch Data	Session End
User Selection	V	N/A	I	N/A
Prerequisites Fulfilled	V	N/A	N/A	N/A
UI Access	V	N/A	V	V
Database Access	V	V	N/A	N/A
Expected Result	Load the User interface	Fetch the Dataset	Display the Result in UI	Return to login page

Step 3: Identify Data Values to Test
The test cases have been identified, reviewed and validated to ensure the accuracy and to identify redundant or missing test cases. Then, once they are approved, the final step is to substitute actual data values for the Is and Vs. Without test data, test cases it cannot be implemented or executed, they are just descriptions of conditions, scenarios, and paths. Therefore, it is necessary to identify actual values to be used in implementing the final tests. Load all the test cases in to matrix A.

TEST METHODS A= {A1, 1, $A_{1,2}$ …$A_{i,j}$}.

4.3 Performing PB-Art Testing

```
1.  Initialize matrix PA.
2.  for each test case in A
3.  generate test block in to row group  n
4.  partition row group n into column group m
    with test methods
5.  for each n X m groups
6.  mark each test methods with states i and j //
    i for non tested, j for tested method.
7.  Add to PA matrix
8.  Execute PerTest(PA)
9.  Output test cases with test patterns
10. end for
11. end for
12. PerTest(PA)
13. for each n X m groups
14. get the mid value
15. perform test for the test case
```

Figure 4. Algorithm2 PB-Art Testing

List partition of a set PA is a division of A into non-overlapping and non-empty parts or blocks that cover all of PA. More formally, these blocks are both collectively exhaustive and mutually exclusive with respect to the set being partitioned.

Given a PA set of testable methods, let be a matrix $PA_{i,j}$ where is a pre-test estimate of the probability of encountering a failure while executing the code to achieve transition from state-i to state-j, while traversing of (i,j) it may be used to automatically partition the test cases of all possible uses .

TEST METHODS PA= {PA1, 1, $PA_{1,2}$ … $PA_{i,j}$}

The list PA = {PA1, 1, $PA_{1,\ 2}$ …… $PA_{i,j}$} is made more precise for an *n* by *m* matrix *M* by partitioning *n* into a collection test block rowgroups, and then partitioning *m* into a collection test method colgroups. The original matrix is then considered as the total of the test method groups, in the sense that the (*i*,*j*) entry of the original matrix corresponds in a one to one and onto way to some (*s*,*t*) offset entry of some (*x* ,*y*), where x € colgroups and y € colgroups.

$$PA = \left\{ \begin{matrix} PA(1,1) & PA(1,2) & \dots & \dots & PA(1,n) \\ PA(2,1) & PA(2,2) & \dots & \dots & PA(1,n-1) \\ \dots & \dots & \dots & \dots & \dots \\ PA(n-1,1) & \dots & \dots & PA(2,1) & PA(2,1) \end{matrix} \right\}$$

4.4 Performance Evaluation

The experiments are executed to investigate the failure-detection capabilities as well as the cost effectiveness of PB-ART on the continuous testing domain and compare them with other existing methods.

The P-measure and the E-measure are most commonly used measures to evaluate testing effectiveness. The P-measure is defined as the probability that at least one failure is detected with a specified test set. The E-measure denotes the expected number of failures detected by the test set. Both measures are evaluated under the predefined set of test cases. When evaluating P-measure and E-measure, we need to prepare the set of test cases without any test execution.

On the other hand, the dynamic testing strategies like ART and our methods utilize the test outcomes to generate subsequent test cases. In other words, we cannot make the set of test cases before executing test cases.

Hence the P- and E-measures are not appropriate to compare dynamic testing strategies. The F-measure as an alternative measure to evaluate performance of dynamic testing strategies, which is defined as the number of test cases needed to detect the first failure. The research work also applies the F-measure to compare our proposed methods with existing ART testing strategies.

5. RESULT EVALUATION

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 the proposed work Fart and Fpbart are used to denote the F-measures for the adaptive random testing and Partitioned block random testing respectively.

Table III: F-measure Values of Fart and Fpbart

Program ID	Functions	Test cases	A	PA	Fart	Fpbart
P1	15	15	2*10	2*10	557.96	538.43
P2	24	24	3*8	3*8	636.19	612.83
P3	32	32	4*5	4*5	661.78	640.23
P4	43	43	9*3	9*3	802.17	792.62
P5	56	56	6*9	6*9	1230.76	1218.94

The empirical study uses a set of 5 error-seeded programs. They are all published programs which are written in C# with program sizes ranging from 30 to 200 statements Table 3 lists the details of the failure rate, type and number of seeded errors for each program.

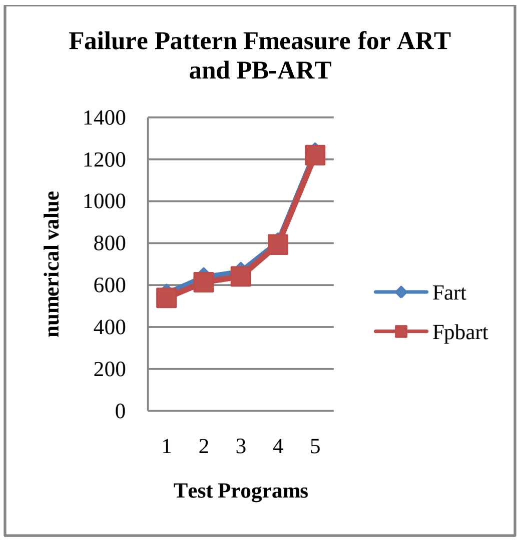

Figure 5. F-measure values of the ART and PB-ART

Figure 5 shows the evaluation result of Adaptive random testing and partitioned block adaptive random testing techniques using F-measure. Experiment carried out using the example programs with test programs. Y axis shows the F-measure numerical values and x axis shows the test programs used. Results show PB-ART outperforms ART technique in F-measure experimental values.

6. CONCLUSION

It is a critical problem in the field of software testing to generate test data with high fault-revealing capability. Chen et al proposed an improved strategy adaptive random testing to overcome this shortage. Besides, it can infer reliability and statistical estimates. 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.

Based on this intuition, the research work proposes a modified version of random testing called Partitioned Block based Adaptive Random Testing. An empirical analysis of 5 published Programs has shown that Partitioned Block Adaptive Random Testing outperforms Adaptive Random Testing significantly for most of the cases.

The experimental results have been providing evidences that the intuition of spreading test cases more evenly within the input space is potentially very useful. Nevertheless, there are a number of issues of Partitioned Block based Adaptive Random Testing that need to be considered, such as various criteria of evenly spreading of test cases ways of defining the candidate sets. It anticipates that analysis of these issues would further improve the effectiveness of Partitioned Block based Adaptive Random Testing.

7. REFERENCES

[1] S. Sarala, “Defects Detection in Imperative Language and C# Applications– Towards Evaluation Approach”, Proceedings of the International Multi Conference of Engineers and Computer Scientists, Vol , pp. 940-944, 2008.

[2] S. Sarala, S.Valli, “A Tool to Automatically Detect Defects in C++Programs”, 7th international conference

on information technology, Springer-Verlag, Vol. 3356, pp. 302-314, 2005.

[3] T.Y. Chen, H. Leung, and I.K. Mak, "Adaptive Random Testing", Springer-Verlag, Vol. 3321, pp. 320–329, 2005.

[4] T. Y. Chen, D. H. Huang, and Z. Q. Zhou, "Adaptive random testing through iterative partitioning", in Proceedings of the 11th Ada-Europe International Conference on Reliable Software Technologies, pp. 155-166, 2006.

[5] S.R.S. Souza, S.R. Vergilio, P.S.L. Souza, A.S. Simao, A.C. Hausen," Structural testing criteria for message-passing parallel programs", Journal of Concurrency and Computation Practice and Experience, Elsevier Publication, pp. 1893–1916, 2008.

[6] Arnaud Gotlieb, Matthieu Petit , "A uniform random test data generator for path testing", Journal of Systems and Software, Elsevier Publication, Vol.83 , pp. 2618–2626,2010.

[7] TsongYueh Chen, Fei-ChingKuoHuai Liu "Enhancing Adaptive Random Testing through Partitioning by Edge and Centre", Proceedings of the 18th Australian Software Engineering Conference IEEE, 2007.

[8] K.-K. Lau, R. Banach, "Adaptive Random Testing by Bisection with Restriction", 7th international conference on formal engineering methods, Springer-Verlag, pp. 251–263, 2005.

[9] W. Grieskamp, C. Weise, "Adaptive Random Testing by Bisection and Localization", 5th international workshop on Formal Approaches to Software Testing, Springer-Verlag, pp. 72–86, 2006.

[10] Borislav Nikolik, "Test Diversity", Journal of Information and software Technology, Elsevier Publications, Vol. 48, pp. 1038-1094, 2006.

[11] Korosh Koochekian Sabor, Mehran Mohsenzadeh, "Adaptive Random Testing Through Dynamic Partitioning By Localization with Distance and Enlarged Input Domain", International Journal of Innovative Technology and Exploring Engineering, Elsevier Publications, ISSN: 2278-3075, Volume-1, Issue-6, 2012.

[12] K. Sayre, J.H. Poore, "Partition testing with usage models", Journal of Information and Software Technology, Elsevier Publications, Vol. 42, pp. 845–850, 2000.

[13] M. Popovic , I. Basicevic, "Test case generation for the task tree type of architecture", Journal of Information and Software Technology, Elsevier Publications Vol. 52, pp. 697–706 , 2010.

[14] Saswat Anand, Edmund K. Burke, Tsong Yueh Chen, John Clark, Myra B. Cohen, Wolfgang Grieskamp, Mark Harman, Mary Jean Harrold, Phil McMinn, "An orchestrated survey of methodologies for automated software test case Generation", Journal of Systems and Software, Elsevier Publications, Vol. 86, pp. 1978-2001, 2013.

[15] Kulvinder singh, rakesh kumar, "Effective Test Case Generation Using Antirandom software Testing", International Journal of Engineering Science and Technology, Elsevier Publications, Vol. 2, pp. 6016-6021, 2010.