Partitioned Block based on Adaptive Random Testing

D. Indhumathi\textsuperscript{a}, S. Sarala\textsuperscript{b}\textsuperscript{*}

\textsuperscript{a}Research scholar, Department of Information Technology, Bharathiar University, Coimbatore, India.
\textsuperscript{b}Assistant Professor, Bharathiar University, Coimbatore, India.

Abstract

Adaptive Random Testing has been proposed towards partitioned block based approach on failures in test cases where the test cases are grouped into neighboring regions. ART algorithm has been described as an alternative for random testing which uses the average number of test case executions to identify failures. The research explores a new ART method using the Partitioned-Block-ART algorithm to reduce the fixed cost in test case generation. The input domain has partitioned into equal cells to achieve successful test cases and coarsely classified into three types such as block, strip and point patterns. The proposed algorithm focuses on block based testing to identify the failures and significantly reduces the time complexity.

Keywords: Random testing, Test cases, Failure pattern, Input domain, Fault detection.

1. Introduction

Software testing defines to detect as many errors as possible with minimum cost. Testing is not restrained only to the detection of error it also assists with the cost of the functional properties of the software. A part of software can be tested to increase the assurance by exposing potential flaws or deviations from the user’s requirements [3].

Testing plays a vital role in software quality improvement [2]. It can be used for software reliability improvement by detecting and removing defects from the software under test [8]. A successful test should expose the occurrence of bugs to a certain extent than proving the working of the program [4]. The author have developed the fuzzy system to generate random data streams to test programs in several versions of the UNIX system [5]. It has been reported that 24% to 33% of the programs tested fails on valid inputs that are randomly generated.

Anti-random testing has been proposed that the first test case be selected randomly, but ART well preserves the randomness since all test cases in ART are randomly selected. ART is based on the observation that failure-causing inputs from different failure patterns [1]. The Research work is based on block pattern, where the failure-causing inputs are clustered in one or a few region. Mostly software models are based on the notion of correctness, i.e. the executed result is “successful” or “failed [15].”

The basics of ART is to uniformly spread test cases, as the concept of “even spread” can be implemented in different ways, several ART methods in terms of algorithms used in the existing work. Distance-based ART (D-ART) [10, 14] and Restriction-based ART (R-ART) [7] are the attempts, which have extensively improved the fault-detection capability of RT. Adaptive Random Testing [13] has been intended to identify common failure patterns than Random Testing.

These methods, however, require additional computational overhead in generating test cases. To decrease the operating cost while retaining the high fault-detection capability, in this research work proposes a new ART method, namely Adaptive Random Testing through Portioned Block (PB-ART) to consume time and cost by developed algorithm. ART is used in terms of finding the number of test cases necessary to detect the first failure [6] [12].

\textsuperscript{*} Corresponding author. E-mail: sriohmau@yahoo.co.in
2. Related Work

The two basic problems in the software industry include the high development cost and the poor qualities of software which have no significant improvement [11]. In this section, D-ART and R-ART methods are briefly reviewed with a focus on time complexity [14]. In this process, the computation and comparison of the distances between test case candidates and all the successful test cases are the main source of computational expenditure.

Mirror Adaptive Random Testing [5] is used to reduce the operating cost. In M-ART, the input domain is first partitioned into equally sized disjoint sub domains, one can be denoted as the original sub domain, and the others are mirror sub domains. D-ART and R-ART is only applied in the original sub domain, and simple mirror functions are used to map the test cases into other sub domains. Empirical study expresses the fault-detection capability of M-ART which is similar to D-ART and R-ART [9]. Nevertheless, the time complexity of M-ART is still quadratic and the numbers of sub domains (w) are being less to ensure randomness.

A uniform random test data generator is derived from the maximum subdomain with constraint programming technique [9]. Divide and conquer algorithm is used to improve the fault-revealing capabilities to build an efficient uniform sequence of test data for a selected path.

Adaptive random testing is to improve the failure detection of random testing which combines random candidate selection with a filtering process. Most of the test cases never reveal failure, so for ART is developed to detect the failure.

Dynamic Partitioning [10] reduces 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 are ART by Random Partitioning and ART by Bisection (B-ART). The input domain in the RP-ART partitions can be used to divide a region by drawing straight lines which is perpendicular to each other crosses at the most recently executed test cases. The sub domains have the largest size to generate the next retrieved test case.

B-ART partitioned the input domain into sub domains of equal size followed by a sub domain might be chosen that does not contains any successful test case as the region to generate the subsequent test case. In case all sub domains contain successful test cases then each sub domain will be subdivided into halves and the testing process is repeated until a failure is detected.

The correlation between test case distribution and failure detection capability has motivated to develop some new ART algorithms, which apply the distribution metrics. In the context of distribution metrics the failure detection capability of ART is stimulated by using distribution metrics mainly discrepancy and spreading which has taken as criteria for the test case selection process.

3. Methodology

The research work proposes a new ART method, namely ART through Partitioned Block Based Adaptive Random Testing. It uses partitioning to identify a test case generation region, where inputs have higher probability of being far apart from all successful test cases.

Predictably, partitioning is a strategy to group elements having similar behaviours in the same sub domain. If such a test case generation region cannot be identified under current partitioning scheme. The input domain will be repartitioned using a finer partitioning scheme. Since PBART does not require the generation of extra candidates and avoids the distance computations and comparisons. Fig 1 which expresses the system flow model inside the testing system. The input is considered as a .net solution file, which is loaded and manifest file of the .net project is read to decode the methods. Finally test cases are generated and executed with all possible test case.

Fig. 2 explains the algorithm for finding the failure and then generates the test data. After finding the failure region a new test input is given.

3.1. Performing PB-Art Testing

Blocks are both collectively exhaustive and mutually exclusive with respect to the set being partitioned. Given a
set of testable methods, let be a matrix $A_{ij}$ where is a pre-test estimate of the probability of encountering a failure.

Stage 1: Initialization
1. Block List=$\emptyset$;
2. set the block as curReg; //curReg represents the current block needed to be partitioned
3. tempP = generateRandPoint (curReg.ll, curReg.ur);

Stage 2: Test Data Generation
1. While true do
2. $\text{Index} =$ findMaxBlock (blockList); //find the max-area block in blockList
3. curReg= blockList.get (index);
4. blockList.remove (Index); //remove the max-area block from blockList
5. if curReg doesn’t contain an existing test input then
6. generate a new test input using function generate RandPoint (curReg.ll, curReg.ur, CurRegn), denote it as $P_1$;
7. if $P_1$ hits the failure-causing block then
8. break;
9. end if
10. end if
11. endwhile

Fig.2. Algorithm for Finding Performance of PB-Art Testing

$$
\begin{bmatrix}
A_{0,1} & A_{0,2} & \cdots & A_{0,n-1} & A_{0,n} \\
A_{1,0} & A_{1,1} & \cdots & A_{1,n-1} & A_{1,n} \\
\vdots & \vdots & & \vdots & \vdots \\
A_{m,0} & A_{m,1} & \cdots & A_{m,n-1} & A_{m,n}
\end{bmatrix}
$$

Fig.3. Test Case Blocks Separated for Performing the Adaptive Random Testing in PB-ART

While executing the code to achieve transition from state-i test method to state-j test succeeds method, while traversing of i,j it automatically partitions all possible test cases automatically.

TEST METHODS $A= \{A_1, A_{1,2}, \ldots, A_{m}\}$. The list $A = \{A_1, A_{1,2}, \ldots, A_{m}\}$ in fig.3 is made more precise for an $n \times m$ matrix $M$. By partitioning $n$ into a collection test block rowgroups, and then partitioning $m$ into a collection of test method colgroups. The original matrix have been considered as the total of the test method groups in the form of $(i,j)$ entry of the original matrix corresponds in a 1-to-1 and onto way to some $(s,t)$ offset entry of some $(x,y)$ where $x$ belongs to rowgroups and $y$ belongs to colgroups.

4. Implementation

A good partitioning strategy results in blocks with different proportions of failures. The greatest gain in precision is realized when all blocks of the partition are homogeneous tests within a block either fail or all of the tests succeed.

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

Step-1: Add the whole input domain into the Block list $L$.
Step-2: Select the max-area block from $L$, denote it as $curReg$, and remove it from $L$.
Step-3: If there are no previous test inputs in $curReg$, a new input point should be randomly generated in this block.
Step-4: $curReg$ is denoted as $P_1$.
Step-5: $P_1$ finds the failure causing block interrupt the process
Step-6: Execute the results

Fig.4. Partitioning the Current Maximum Area Block

The adaptive random test data generation algorithm, as based on two-point partitioning can be described in the form of pseudo-code. The figure 4 pseudo-codes, function generateRandPoint can randomly generate a test input 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 and upper-right point. The function findMaxRegion returns the index of the max-area region in region List. There are several kinds of operations can be invoked by object region List such as get $()$, remove $()$ and so on.

4.1 Loading The .Net Solution File

Test cases choose the collection of code mostly from texts relating programming in the C# language. Such texts not only tend to provide a large body of code in a single location, but can also be expected to use a good range of the language’s features in the process of explaining them. This contrast with sample applications, which may only give attention to a subset of the language features, either the concentration on a particular domain of application, or because of the coding method of the author. Manifest file of the c# solution file is read and the method is parsed from the manifest information.

The figure 5 shows the difference in time from loading a testing project from code to manifest file reading. Loading from code requires the test method pattern matching whereas loading from the manifest provides file library and compiled information.

Each manifest class provides details about the enclosed class and method details. Since manifest read the pre-
compiled library information. It is faster compared to loading from code.

![Graph showing comparison between solution code loading and manifest loading](image)

**Fig.5. Solution Code loading VS Manifest Loading**

### 4.2 Test case generation

For generating test case PBART creates a testing library which uses C# attributes to perform testing and the library contains a Test Fixture. This is accomplished by creating a class and assigning it as [Test Fixture] attribute.

The [Test Fixture] attribute takes no parameters and can be applied to classes only. The purpose of the [Test Fixture] attribute is to identify which classes will be running test cases. Upon loading your test get-together, system searches the complete assembly for classes with this custom attribute set. The next step is to create test methods that will run the actual tests case for this work.

### 4.3 Creating Test Methods

The [Test Fixture] contains one or more methods that are flagged with the attribute. This attribute states the outline of particular method in the fixture is to be run during the testing phase. This method should be mutually exclusive of their state and should be designed to not have any side-effects. If you leave side effects, then subsequent test methods may be adversely affected by the leftover state.

The graph in figure 6 provides the time complexity involved in generating the test cases based on information provided by manifest file. X axis shows the test methods generated and Y Axis shows the time taken for generating the test methods. At 5.2 seconds it has generated around 110 test cases.

![Graph showing test case generation](image)

**Fig.6. Test Case Generation Graph Map**

### 5. Experimental Results

Test cases choose a range of code, mostly from texts describes programming in the C# language. Such texts not only tends to provide a large body of code in a single location, but can also be expected to use a good range of the language’s features in the process of explaining them. In Table1 the loaded C# programs are executed with some methods to find the results.

<table>
<thead>
<tr>
<th>Test id</th>
<th>Loading Exe File</th>
<th>Test Methods</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Qpaid.exe</td>
<td>Button1Test Click Function</td>
<td>Pass</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TextChangeClick</td>
<td>Pass</td>
</tr>
<tr>
<td>2</td>
<td>Cluster.exe</td>
<td>MenuClickAlert TextBoxData Load</td>
<td>Pass</td>
</tr>
<tr>
<td>3</td>
<td>BankApp.exe</td>
<td>NewFormOpen TestFunctionUser MenuItemExitClick</td>
<td>Fail</td>
</tr>
<tr>
<td>4</td>
<td>Intrusion.exe</td>
<td>TextBoxValidation</td>
<td>Pass</td>
</tr>
</tbody>
</table>

**Table 1: Tested Files And Results**

462
6. Conclusion

The proposed method focuses the process of partitioning the input to identify the current region, which causes failure while executing the code. 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 the automated testing tools due to its merits 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 does not give high-quality. Hence the developed algorithms significantly reduce the time complexity and cost.

Partitioned block based adaptive random testing can randomly generate a test input in the current region and selects the current max-area region as partition object. Then the partition can be iteratively performed until the potential faults are found.

References


