IMPROVISING THE QUALITY OF ARPT USING GREFENSTETTE METHOD

M. N. POORNA KISHORE

M. Tech. Student, JNTUA CEA, Ananthapuramu, India
E-mail: mnpkishore531@gmail.com

Abstract— Adaptive and Random Partition Testing (ARPT) is a software testing strategy that is effective in defect detection. Its simplicity in defect detection makes it efficient with respect to time. Any testing strategy has to be initialized with the parameters like length of the testing segment, and when to change the testing strategy before the testing commences. These parameters do not get any values/inputs from the tester manually. These parameters have to be declared with the testing itself. Analyzing the ARPT by sensitivity analysis states that, effectiveness of ARPT can be increased by optimizing the parameters. Optimization of parameters helps to improve overall quality of ARPT detecting the maximum number of defects with optimal count of test cases. In this paper, limitation of ARPT is solved by using grefenstette method.

Index Terms— Adaptive and Random Partition Testing, grefenstette method, sensitivity analysis, software testing.

I. INTRODUCTION

Software Testing (ST) is a major approach in assuring the quality of software. High efficient software will be engineered, only when it is tested in all of its development phases. ST can be either a black box (functional) or a white box (structural) approach. Random testing is used for selection of test cases with a large input domain. A part of code to be tested termed as a testing segment. Testing segment length is considered as one of the parameters in the testing domain. Length of the Testing segment is not fixed for all the testing techniques. They are variable with respect to the techniques available and their usage.

ARPT is a black box approach. APRT is effective in detection of defects. APRT is a feedback based testing technique, through which it has to adapt itself for the future testing process of any software. ARPT is a hybrid approach of Adaptive Testing (AT) and Random Partition Testing (RPT) [1]. The combination of AT and RPT testing techniques helps in balancing the testing effectiveness and defect detection. As ARPT is a black box approach, it compares the resulted output with the expected output. A Heuristic technique was implemented in order to define the length of the segment to be tested in ARPT. By sensitivity analysis on ARPT, it is said that, the efficiency of ARPT can be improved by optimizing the parameters.

Grefenstte method is one of the parameter optimizing technique. In grefenstette method, a feedback process initiates and sends the result back to the observer to adapt itself to the corresponding environment [2]. By integrating the grefenstette method to the ARPT technique, we improve the quality of ARPT in terms of detecting the crest number of defects with optimal number of test cases. Paper is organized as follows: An overview of the related work is discussed in II section. Section III describes the integration of ARPT with Grefenstette. The experimental design, it's working and results are discussed in section III. Summary on conclusion is in section V.

II. RELATED WORK

A. ARPT

The motivation of ARPT is to detect maximum number of defects with minimum number of test cases. This is done by aggregating both AT and RPT techniques so that the computation complexity of AT is reduced by introduction of RPT in the testing process without affecting the efficiency of defect detection [1]. ARPT utilizes AT and RPT in an alternate manner to enhance its strength. Parameters like state of the testing process "*t*", signal $S_F(t)$ to indicate the detection of error in software under test, and the length of the testing segment "*m"* are to initialized before the testing process begins. Parameter for length of the testing segment is determined by probability distribution on feedback history. According to Junpeng *et al.* length of the segment that is to be tested using ARPT is not of a fixed length [1]. The lengths vary according to the number defects found in its previous testing process.

1. t = 0,
2. n = k(t) + l(t)
3. t = t+1,
4. $S_F(t) = 1$, if any defect is detected.

B. CODE COVERAGE OF ARPT

ARPT is effective in terms of both testing effectiveness and defect detection [1]. More the rate of code coverage, leads to increase the capability of failure detection [3]. Inspecting the whole code and detecting the defects will surely ensure the quality of the software under test.

Types of code coverage's are statement coverage, function coverage, branch coverage, decision coverage, etc. Branch coverage ensures us to reach all the possible branches from a decision point and each branch is run at least once.

Code coverage capability of AT and RPT is high and stable [4]. The aggregated model, ARPT is engineered by considering the advantages of the AT & RPT.

C. GREFENSTETTE METHOD

According to grefenstette, optimization of a process can be achieved in two ways. Optimization is done either by choosing pack of optimization algorithms that are suitable for the process or by tuning the parameters that are initialized before the process. A strategy tuner is being used in this work to optimize the parameters that are initialized.

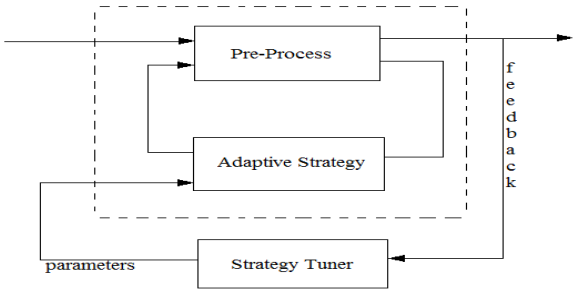

Fig: **Grefenstette Adaptive Strategy**

Grefenstette method works on branch coverage concept and assures us high code coverage. Strategy tuner helps in covering the whole code by tuning itself to the environment.

III. ARPT with GREFENSTETTE

Main aim of integrating ARPT with Grefenstette method is to enhance the quality of ARPT with respect to detecting maximum defects with minimum test cases. Along with that, work aims at balancing testing effectiveness and defect detection. ARPT is a black box testing approach which aims at verifying the syntax but not the working of the code as in the process of white box testing.

As mentioned earlier, grefenstette method tunes the pre-initialized parameters as per the requirements and is forwarded to testing. For this tuning of parameters, branch coverage is being used.

A. Branch Coverage

Branch coverage is method of testing, which aims to execute all the possible branches at least once from each decision, thereby assuring that all the code is executed [5]. For better testing results, covering all the code is very operative.

Branch coverage is a pre-process technique in the ARPT testing process. Before stepping into the adaptive testing, all process work is done and then the tuning of those parameters is done. As ARPT is a feedback based testing technique, feedbacks from all the results of earlier testing process has to be collected. Using the collected information, the strategy tuner tunes the parameters and sends the inputs to the adaptive testing process.

B. Strategy Tuner

Although all the pre-process work is done before the testing, tuning of parameters is essential for effective testing results. This strategy tuner acts as a cleansing phase or a filtration phase. Though we have many parameters initialized earlier, this phase filters the parameters that are needed for testing and pass it to the actual testing process.

C. Adaptive Testing

Adaptive Testing (AT) has key role in ARPT. It helps in verifying the adaptability of the code to be tested for a particular environment. These parameters of the required environment must not be altered. In the process of ARPT, along with AT, Random Partition Testing (RPT) is also done. Both are done in alternate manners. RPT helps in partition of code and testing it in a random way.

D. Parameter setting in ARPT

ARPT deals with 2 parameters in the testing process.

1. Size of code to be tested.
2. Point of swapping between AT and RPT

Both the testing techniques i.e., AT and RPT are essential in ARPT. A Heuristic method implements the code and defines the size of code to be tested. Size is determined by the feedback history. Length of code to be tested need not be the same for all testing process. It varies depending on the environmental parameters.

Each and every partitioned code, has to undergo AT and RPT. For this, switching between the testing strategies has to be done. If a defect is found in code while testing, a signal ($S_F(t)$) is passed as an acknowledgement of defect.

IV. EXPERIMENTAL DESIGN

As ARPT is a black box testing technique, testing the code for identifying syntactical errors are aimed. As the whole code cannot be tested at a time, the code has to be partitioned. For partitioning of code we implement the branch coverage technique. Using the branch coverage technique, we part the code for every decision or a condition.

```
Condition1
{
        Condition2
        {
        …..…
        }
}
```

Sample code of a program

Any sort of program written in java, contains the decisions or conditions. Partition of code can be done in many ways. Using branch coverage in this work of ARPT with grefenstette, code of the whole program to be tested is covered. The work is done as follows.

1. Select a file to test.
2. Part the code.
3. ARPT

A file to test is the input of the whole testing process. For large size programs, the whole and the single file is given as input. Selection of file to be tested is taken from the system directory. The file is taken as the input. Selection of file that is to be tested is made easy while choosing from the directory. This is common aspect in each and every testing process.

A. Partition of code

Partitioning of code is the major aspect with respect to the length of code that is to be tested. In the process of ARPT with grefenstette, partition of code is dealt by applying branch coverage using grefenstette method. Using branch coverage, all the code id covered and part the code according to the decisions and conditions. Partition of code completely is done in two stages: splitting and configuration of code.

Splitting of code

Splitting of code is a stage where splitting of code is done according to the procedures or functions or methods.

```
void Test4 {
        int f1=0,g1=1;
        if (f1 == g1) {
        g1++;
        }
        else if (f1 == g1) {
        f1++;
        }
else {
g1--;
}
}
```

Sample split code

After the splitting process is done we get a list of all functions starting from function ‘1’ to function ‘n’. n represents the total number of functions in the whole code. A sample code that is split after the process of splitting is as the above one. In the same way we get as many as functions. Each function may consist any number of decisions and conditions. After the process of splitting is done, configuration of split code proceeds.

Configuration of code

Configuration of code states that arrangement code in an order. As the result of splitting the code are functions, configuring the functions favoring to the best results of testing has to be done, and that is done by configuration of code. Irrespective of the main class used in programming, configuration of code concentrates on conditional statements. All other unnecessary data is not considered for testing. Sample code after configuring the split code is as below:

```
if (a == b) {
a++;
}
else if (a == b) {
b++;
}
else {
b--;
}
```

Sample configured code

As configuration of code states that the code has testing phase. After the configuration of the code is done, the process of testing begins.

Length of the testing segment

Length of the testing segment is not fixed to some part. In ARPT the length varies with respect to the feedback history. If no defect is detected in the specified length (that is determined from feedback history), then the length of the testing segment for the next iteration gets doubled in actual ARPT. In the work of ARPT with grefenstette, length of the testing segment is different and the length to test is fixed to some conditional statements. Length of the inner code in conditional statements is not taken into consideration.

The syntactical errors are found by verifying the number of parenthesis for each and every conditional statement. Number of parenthesis must be always stable i.e., zero. Outputs depend on the number of parenthesis.

```
If (count ==0) {
“no errors”
}
Else {
“errors exist”
}
```

Whenever the count is not equal to zero, the value will be in terms of ‘-’ 0r ‘+’. On the process of testing after configuration, each time when the “{” is encountered, the count is incremented by ‘1’. In the same way, When the “}” is encountered by the testing process, the count is decreased by ‘1’. Value of count ranges in [-n, n]. If the value is in terms of +n, then an acknowledgement of error is reported and it states that number of parenthesis are not stable and an extra ‘{’ in present in the code. Likely, if the error is

acknowledged as '-n', then an extra '}' is present in the code.

B. Results

To evaluate and compare the effectiveness of ARPT and ARPT with Grefenstette, a set of tests are conducted on various program files. These files are tested using ARPT with grefenstette, and is compared with the results of ARPT, that are stated in the work of Jupeng *et.al*..

By using the branch coverage technique, and by implementing the grefenstette method, the capacity of detecting defects using ARPT if increased.

CONCLUSIONS AND FUTURE WORK

Testing is a major phase in the software development life cycle. In recent times, AGILE methodology is a buzz word. In agile methodology, and few of the development methodologies, testing in every phase are giving the best results. As whole code cannot be engineered at a time and only few lines of code is developed, this ARPT with grefenstette method produces optimal results when compared to other black box testing strategies.

The future work extends to work with other genetic algorithms that help in enhancing the quality of ARPT with genetic algorithms.

REFERENCES

[1]. Junpeng Lv, Hai Hu, Kai-Yuan Cai, and Tsong Yueh Chen, Adaptive and Random Partition Software Testing, IEEE transactions on systems, man, and cybernetics: systems, vol. 44, no. 12, December 2014.

[2]. JOHN J. GREFENSTETTE, Optimization of Control Parameters for Genetic Algorithms, IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS, VOL. SMC-16, NO, 1, JANUARY/FEBRUARY 1986.

[3]. Chen, T, Kuo, F, Liu, H and Wong, E 2013, 'Code coverage of adaptive random testing', IEEE Transactions on Reliability, vol. 62, no. 1, pp. 226-237.

[4]. Feng Ye, Chang Liu, Hai Hu, On the Computational Complexity of Parameter Estimation in Adaptive Testing Strategies, 15th IEEE Pacific Rim International Symposium on Dependable Computing, 2009.

[5]. K. Devika Rani Dhivya, Dr. V. S. Meenakshi, A Comparative Study on Adaptive and Random Testing Technique, International Journal of Advanced Research in Computer Science and Software Engineering, Volume 6, Issue 2, February 2016.

★★★