On the Effectiveness of Genetic Operators in Adaptive Random Testing

S.P. Ng*, F.C. Kuo, R.G. Merkel, T.Y. Chen
School of Information Technology
Swinburne University of Technology
Hawthorn, Victoria 3122
AUSTRALIA

Email: {sng, diana_kuo, rmerkel, tychen}@it.swin.edu.au
Phone: +61-3-9214-8666
Fax: +61-3-9818-0823

ABSTRACT

Adaptive random testing (ART) has recently been found to be an effective way to improve the performance of random testing by selecting widespread test cases from the input domain. In this paper, we outline how the selection, mutation and crossover operators of the genetic algorithm (GA) can be used in ART, and investigate experimentally which of these is the most appropriate operation to serve for the purpose of widespread as required in performing ART.

KEYWORDS

Adaptive random testing, Black box testing, Genetic algorithm, Random testing, Software testing, Test case selection

1. INTRODUCTION

Software testing has been shown to be the most systematic and pragmatic approach towards improving software quality. With increasing customer expectation and ever-growing system complexity, testing is becoming more important and also more professional in the software development life cycle. In the past, many techniques have been introduced to test software applications [1, 11,13]. Testers can employ black box or white box approach or often both to select test cases from the input domain. Popular black box techniques include random testing [10], category-partition method [12] and classification-tree method [4, 8], while common white box techniques include control-flow coverage and data-flow coverage methods.

Despite of its simplicity in concept, random testing still remains as one of the most popular black box techniques. Other advantages such as easy implementation and automation, and its ability to infer software reliability are the major reasons for the wide adoption of random testing in the software testing industry.

Adaptive random testing is a recently developed method to improve random testing (for example, see [3, 5]). It was designed to improve the fault-detecting effectiveness of random testing in situations where the failure-causing region is contiguous such as those with block or strip patterns [2].

A block pattern of failure-causing region refers to the case when the failure-causing inputs are clustered within a closed region. As an example, consider the following program segment (written in C++) with a typographical error in one of the assignment statement of Z:

```
cin >> X >> Y;
if ((X >= 10 and X <= 11) && (Y >= 10 and Y <= 11))
then Z = X + Y;
//the correct statement should be "then Z = X – Y"
else
Z = 10;
cout << Z;
```

The pattern of the failure-causing inputs of this program is a block pattern as sketched in Figure 1.

A strip pattern of failure-causing region refers to those when the failure-causing inputs form the shape of a narrow strip in the input domain.
An example of this pattern can be observed in the following program segment (in C++) with the condition in the ‘if’ statement coded incorrectly:

```cpp
cin >> X >> Y;
if (X - Y > 11)
    // the correct statement should be "if (X - Y > 10)"
    then Z = X + Y;
else
    Z = 10;
cout << Z;
```

The pattern of failure-causing inputs for this program is a strip pattern as sketched in Figure 2.

A basic version of ART can be described as follows. Suppose T = \{T_1, T_2, \ldots, T_m\} is the set of executed test cases. Then, randomly generate a set of candidate test cases C = \{C_1, C_2, \ldots, C_n\} from the input domain. Determine the candidate C_k of C, such that C_k is furthest away from the elements of T, as the next test case to be executed. If C_k does not reveal any failure, add it into the set T and the process is repeated until a failure is detected or testing terminates according to some pre-defined criteria, whichever earlier.

Previous empirical studies have shown that ART can significantly outperform random testing when block or strip patterns of failures occur [5]. Chen et al. investigated the performance of ART by seeding errors into some published programs and examining the average number of test cases required to detect a failure (a testing metric known as the F-measure). In their simulation studies, it was found that for most programs used in the experiment, ART performed significantly better than random testing, with the average F-measure of ART being approximately 60% of the average F-measure of random testing.

Genetic algorithm has been shown to be a well-established iterative approximation method for finding near-optimal solutions to some hard-to-solve problems (for example see [7, 9]), such as the travelling salesman problems. As there are extensive overheads in deciding which candidate is furthest away from all the already executed test cases, this study attempts to use GA to achieve the intelligent widespread of test cases.

For GA, an initial “generation” of solutions to a problem is normally created by choosing available candidates randomly or according to some intuitions. Obviously, there is no surprise that these initial solutions may be quite poor. Nevertheless, a new pool of candidates is subsequently generated from this initial pool by application of the following three operators:

- **Crossover**: in which pairs of members from the previous generation are combined by some processes to form an element of the new candidate set;
- **Mutation**: in which a member from the previous generation is altered by modifying a part of itself to form an element in the new candidate set; and
- **Selection**: in which a member from the previous generation is selected to remain in the new candidate set.

Each member in the candidate pool is then evaluated using a fitness function. Based on their relative fitness, the best candidates are retained, the rest is discarded and the process iterates until a sufficiently good solution is eventually found.

In this paper, we examine methods for performing these basic operations of GA in the context of ART. We also present an empirical analysis of the effectiveness of these basic operations.

The rest of this paper is as follows. Section 2 explains our method of applying the three basic genetic operators (mutation, crossover and selection) in ART. Section 3 describes our simulation study. Section 4 presents the results of our simulation study and concludes the paper.
2. GENETIC ALGORITHM IN THE CONTEXT OF ART

2.1 Test case and failure region representation

In this investigation, we only considered a program whose inputs are integers. For simplicity, we further assumed that the input domain consisted of two variables, namely Vx, Vy and both were restricted to the range [0, 127]. Hence, every test case can simply be represented as a 14-bit vector, where the leftmost seven bits and the rightmost seven bits contain the binary representation of Vx and Vy respectively. In spite of the simplification, this model is readily extensible to input domains of more than two dimensions.

For all simulations described in this paper, we randomly assign a square region within the input domain as the failure-causing region. Any test case selected that falls within this region is regarded as a program failure (that is, a deviation from the expected performance of the program). A failure-causing region of this nature is a typical example of the block failure pattern [2].

2.2 Genetic-ART

In this section, we describe a method of applying the basic genetic operators in ART, which we name Genetic-ART (hereafter referred to as G-ART). In the proposed G-ART, two sets of program inputs are recorded – the set of inputs already used to test the program (known as the executed set), and a set of as-yet unexecuted potential test inputs (known as the candidate set). While the size of the candidate set is fixed, the executed set grows in size as the algorithm proceeds. Initially, the executed set is empty and the candidate set is filled with inputs randomly chosen from the input domain (with all inputs from the domain equally likely to be chosen as candidates).

Members of the candidate set are evaluated by a fitness function in light of the executed set (see Section 2.3 for details of fitness function) and the candidate with the highest fitness is chosen for test execution. The only exception is in the first iteration when the executed set is empty, all candidates have the same fitness and so in effect the choice is totally random. A test is then carried out using the chosen candidate. If the test detects a failure, the procedure halts. Otherwise, the input is added to the existing executed set and a new set of candidates is then generated by applying the basic genetic operators (selection, mutation, and crossover) to the current candidate set. The proportion of the new set of candidates generated by each method was a specifiable input to the method (so, for instance, we could specify that of the set of 10 candidates, 2 were to be generated by selection, 4 by mutation, and 4 by crossover). The algorithm continues until a failure is detected by a chosen candidate or some other stopping criterion (such as a maximum number of tests) is reached.

In G-ART, a new “solution” (i.e. input to be used as a test case) is produced and used after each iteration of the algorithm. This is different to standard GA practice, in which solutions are only used after completing many generations of evolution. In typical GA applications, only one ultimate solution is required, and a large amount of computer time is expended to obtain that solution. In our case, many are required, and there is no need for an “optimal” solution every time. A good-enough solution, quickly obtained, is preferable. Therefore, we considered it appropriate to use the solutions obtained after each iteration.

Details about the use of the three genetic operators of mutation, crossover and selection to generate a new candidate set will be further discussed in subsequent sections.

2.3 Fitness function

The fitness function employed to select the next test case from the candidate set is identical to that used in the basic version of ART as described in Section 1. It makes use of the Cartesian distance between each candidate test case and its closest already-executed test case [5]:

\[ \text{fitness}(c_j, E) = \min_{i=1, \ldots, m} \{ \text{dist}(c_j, e_i) \} \]

where \( E \) is the executed set of test cases, \( e_i \) is an element of \( E \), \( c_j \) is an element of the candidate set \( C \), \( m \) is the size of \( E \), and \( \text{dist}(p,q) \) denotes the Cartesian distance between two points \( p \) and \( q \) in the input domain.

The chosen candidate is the one such that the above fitness function is maximised with respect to all already executed test cases. In this way, we achieve the notion of widespread as desired in ART.
2.4 Mutation

In our simulations, mutation was performed by simply flipping two arbitrary bits of the 14-bit vector. To ensure a certain degree of randomness in the mutated test cases, the two bits to be flipped in the pattern of each candidate are determined through a pseudo random number generator.

In addition to the one described above, other mutation methods were also explored. Our preliminary studies showed that the above method indeed produces the greatest amount of “disorder” in succeeding generations. Since there are extra overhead of generating a random number to decide which bit to flip, this method may not be the most economical way for mutation. However, in this study, we chose to investigate the optimal technique for each operator.

2.5 Crossover

Various approaches have been attempted to achieve the best crossover performance, but preliminary experiments revealed that most methods resulted in successive generations closely clustered around a limited number of bit patterns. After some trials, a crossover method that showed the least tendency to produce such an undesirable outcome was devised.

In our proposed crossover method, the 14-bit vector of the new candidate produced by two candidates randomly picked from the original pool was made up of three portions: the first and the last portion came from one of the two candidates and the middle portion came from the other. In the generation process, the first or the last portion may be empty.

2.6 Selection

In our experiments, selection was performed based on fitness. Candidate with the highest fitness (relative to the executed set of test cases) is retained as the new candidate test case. This method, though simple, is very commonly used in genetic algorithm applications.

A simulation study was conducted to determine the effects of the basic genetic operators on the failure-detection capabilities of the G-ART algorithm. In this study, the size of the candidate set is kept at 10.

Our study examined G-ART in the case of a 2-dimensional integer domain of size 128 x 128 as previously described, with a randomly chosen square failure-causing region of size 4 x 4. The square shape of the failure-causing region was chosen as it was found to be the one where ART performs better than random testing [5]. For random testing without replacement, the average F-measure is expected to be

\[ F = \frac{128^2}{4^2} = 1024 \]

To study their relative influences, the proportion of the candidates generated by selection, mutation and crossover was varied systematically in our simulation. For each proportion, the same algorithm is applied repetitively until a chosen test case fell into the failure-causing region, and the number of time that the process required (hence the F-measure) was recorded. The experiment is then repeated using a new initial generation of candidate test cases and another new failure-causing region.

The number of experimental trial required was determined by calculating the confidence interval for the mean F-measure. According to the central limit theorem [6], the sample size required to estimate the mean of F-measure with an accuracy range of ± r% and a confidence level of (1-\(\alpha\)) x 100%, where 1-\(\alpha\) is the confidence coefficient, is given by

\[ S = \left( \frac{100 \cdot z \cdot \sigma}{r \cdot \mu} \right)^2 \]

where z is the normal variate of the desired confidence level, \(\mu\) is the population mean and \(\sigma\) is the population standard deviation.

In our simulations, we have chosen a confidence level of 95% and accuracy range of ± 5%. In other words, the process continues until the sample mean of F-measure is accurate within 5% of its value at 95% confidence level. Since \(\mu\) and \(\sigma\) are unknown, they are replaced respectively by the mean (m) and standard deviation (s) of the collected F-measure data. From statistical table, z = 1.96 for 95% confidence. Hence, the simulation program will only stop when the number of trials
reaches the value evaluated by the following equation:

\[ S = \left( \frac{100 - 1.96 \cdot s}{5 \cdot m} \right)^2 \]

The simulation results indicated that our choice of 5% accuracy and 95% confidence are sufficient to stabilise the estimated mean towards its population mean, within reasonable amount of effort and resources.

To examine the relationship between the expected F-measure value and the proportion of candidates generated by the three genetic operators of mutation, selection and crossover, experiments were conducted in the following way: The proportion of candidates generated by one parameter, say mutation, was fixed at a “low” level (chosen to be 20%), and the proportion generated by the other two operators (selection and crossover) was then varied for three different experimental conditions. The same procedure was then repeated with mutation increased to a “medium” level (40%), and then a “high” level (60%). In this manner, the relative influence of the other two operators (selection and crossover) could be compared as the amount of mutation changes.

For any fixed proportion of candidates generated by one genetic operator, say mutation, the candidate proportion generated by the other two operators (selection and crossover) was varied in such a way to provide a larger difference between the ratios of candidates generated by these two operators. For example, in the experiments with proportion of mutated candidates fixed at 20%, the three pairs of candidate proportion generated by selection and crossover given by (20%, 60%), (40%, 40%) and (60%, 20%) were used instead of (30%, 50%), (40%, 40%) and (50%, 30%) because the former set of parameters provided ratios of 1/3 (20:60) and 1 (40:40) while the latter only provided a much smaller ratio difference between 3/5 (30:50) and 1 (40:40).

Similar procedures were carried out for the experiments of fixed selection (varying mutation and crossover) and fixed crossover (varying mutation and selection) separately.

4. DISCUSSION AND CONCLUSION

Table 1 shows the effect on the expected F-measure by varying the candidate proportion generated by selection and crossover, with those by mutation fixed at 20%, 40% and 60%:

<table>
<thead>
<tr>
<th>Mutated</th>
<th>Selected</th>
<th>Crossover</th>
<th>Trials</th>
<th>Expect(F)</th>
<th>Std Error</th>
<th>95% Confidence Int.</th>
</tr>
</thead>
<tbody>
<tr>
<td>20%</td>
<td>20%</td>
<td>60%</td>
<td>1080</td>
<td>703</td>
<td>17.8</td>
<td>668 - 738</td>
</tr>
<tr>
<td>20%</td>
<td>40%</td>
<td>40%</td>
<td>1120</td>
<td>748</td>
<td>19.1</td>
<td>711 - 785</td>
</tr>
<tr>
<td>20%</td>
<td>60%</td>
<td>20%</td>
<td>1220</td>
<td>751</td>
<td>19.1</td>
<td>711 - 785</td>
</tr>
<tr>
<td>40%</td>
<td>10%</td>
<td>50%</td>
<td>980</td>
<td>650</td>
<td>16.5</td>
<td>613 - 683</td>
</tr>
<tr>
<td>40%</td>
<td>30%</td>
<td>30%</td>
<td>900</td>
<td>684</td>
<td>17.3</td>
<td>650 - 718</td>
</tr>
<tr>
<td>40%</td>
<td>50%</td>
<td>10%</td>
<td>900</td>
<td>704</td>
<td>17.9</td>
<td>678 - 741</td>
</tr>
<tr>
<td>60%</td>
<td>10%</td>
<td>30%</td>
<td>980</td>
<td>639</td>
<td>16.2</td>
<td>607 - 671</td>
</tr>
<tr>
<td>60%</td>
<td>20%</td>
<td>20%</td>
<td>980</td>
<td>626</td>
<td>15.9</td>
<td>595 - 657</td>
</tr>
<tr>
<td>60%</td>
<td>30%</td>
<td>10%</td>
<td>1000</td>
<td>675</td>
<td>17.2</td>
<td>642 - 709</td>
</tr>
</tbody>
</table>

From the results shown in Table 1, it can be observed that if the proportion of mutated candidates is fixed, altering the proportion of candidates generated by crossover and selection has no significant effect on the expected F-measure. However, there is an observable trend that increasing the mutation proportion tends to reduce the expected F-measure.

Table 2 shows the effect on the expected F-measure by varying candidate proportion generated by mutation and crossover, with those by selection fixed at 20%, 40% and 60%:

<table>
<thead>
<tr>
<th>Mutated</th>
<th>Selected</th>
<th>Crossover</th>
<th>Trials</th>
<th>Expect(F)</th>
<th>Std Error</th>
<th>95% Confidence Int.</th>
</tr>
</thead>
<tbody>
<tr>
<td>60%</td>
<td>20%</td>
<td>20%</td>
<td>980</td>
<td>626</td>
<td>15.9</td>
<td>595 - 657</td>
</tr>
<tr>
<td>40%</td>
<td>20%</td>
<td>40%</td>
<td>920</td>
<td>670</td>
<td>17.0</td>
<td>637 - 704</td>
</tr>
<tr>
<td>20%</td>
<td>20%</td>
<td>60%</td>
<td>900</td>
<td>720</td>
<td>18.7</td>
<td>666 - 740</td>
</tr>
<tr>
<td>50%</td>
<td>40%</td>
<td>10%</td>
<td>960</td>
<td>663</td>
<td>16.8</td>
<td>630 - 696</td>
</tr>
<tr>
<td>30%</td>
<td>40%</td>
<td>30%</td>
<td>980</td>
<td>674</td>
<td>17.0</td>
<td>642 - 709</td>
</tr>
<tr>
<td>10%</td>
<td>40%</td>
<td>50%</td>
<td>1280</td>
<td>907</td>
<td>23.7</td>
<td>862 - 953</td>
</tr>
<tr>
<td>10%</td>
<td>60%</td>
<td>10%</td>
<td>1060</td>
<td>733</td>
<td>18.0</td>
<td>697 - 770</td>
</tr>
<tr>
<td>20%</td>
<td>60%</td>
<td>20%</td>
<td>1220</td>
<td>751</td>
<td>19.1</td>
<td>713 - 788</td>
</tr>
<tr>
<td>10%</td>
<td>60%</td>
<td>30%</td>
<td>1440</td>
<td>965</td>
<td>24.5</td>
<td>917 - 1013</td>
</tr>
</tbody>
</table>

The results in Table 2 show that, for fixed proportion of selected candidate, increasing the proportion of candidate generated by mutation and decreasing that generated by crossover reduced the expected F-measure. There is also an overall trend that the smaller the proportion of candidates generated by selection, the smaller the expected F-measure.

Table 3 shows the effect on the expected F-measure by varying candidate proportion generated by mutation and selection, with those generated by crossover fixed at 20%, 40% and 60%:
The results in Table 3 show that if the proportion of candidate by crossover is fixed, increasing the proportion of candidates generated by mutation and decreasing that by selection reduced the expected F-measure. In addition, decreasing proportion of candidates generated by crossover tends to slightly reduce the expected F-measure, but not to the extent produced by the selection operator as observed in Table 2.

In view of the above results, we can conclude that amongst the three genetic operators, mutation has outperformed selection and crossover in achieving the desired widespread effect of test cases.

REFERENCES