MULTI-OBJECTIVE OPTIMIZATION FOR TEST DATA GENERATION USING GENETIC ALGORITHM

P. Maragathavalli, S. Kanmani, R. Vishnu Priya, O. Shanmougapiiya, B. Manoranjitham and B. Vindya Lakshmi

Department of Information Technology, Pondicherry Engineering College, Puducherry, India
E-mails: marapriya@pec.edu, kanmani@pec.edu, vprkriya@pec.edu, priyashanu@pec.edu, ranjiniglitters@gmail.com, lucky.vindya@gmail.com

Abstract: This paper presents a object-based evolutionary algorithm (OBEA) to generate test cases for object-oriented programs using search-based techniques. Multi-objective optimization is the process of simultaneously optimizing two or more conflicting objectives subject to certain constraints. The objectives considered in this paper for optimization are maximum branch coverage, minimum execution time. The new genetic operators like rank selection, arithmetic crossover, two point random mutations are attempted in genetic algorithm to improve the performance of generated test cases. The experiment shows that when the number of conditions increases execution time also increases.

Keywords: Genetic algorithm, test data generation, execution time, branch coverage

I. INTRODUCTION

Evolutionary algorithms (EAs) are search methods that take their inspiration from natural selection and survival of the fittest mechanisms [6]. EAs differ from traditional optimization techniques in that they involve a search from a population of solutions, not from a single point. Each iteration in an EA involves a competitive selection that filters the least favorable solutions. The solutions with high fitness are recombined with other solutions by swapping parts of a solution with another. Solutions are also mutated by making a small change to a single element of the solution. EAs are robust optimization methods used for test data generation.

Multi-objective optimization refers to the solution of problems with two or more objectives to be satisfied at the same time. Most real world problems have multiple objectives to achieve. This situation creates a set of problems in Multi-Objective Optimization Problems (MOOP) [1]. A MOOP has a number of objective functions, which are to be minimized or maximized.

MOOP produces a set of solutions which are superior to the rest of the solutions with respect to all objective criteria but are inferior to other solutions in one or more objectives. These solutions are called Pareto Optimal solutions or non-dominated solutions [1][3]. A Pareto optimal set is the mathematical solution to a multi-objectives problem. A solution is Pareto-optimal if no other solution can improve one object function without reducing at least one of the other objectives.

Genetic Algorithms (GAs) are adaptive heuristic search techniques based on the evolutionary techniques of natural selection [2][8], recombination and mutation. The principle behind GAs is that they create and maintain a population of individuals represented by chromosomes population of individuals represented by chromosomes [6]. The input to the algorithm is a set of potential solutions to that problem and a metric called a fitness function allows each candidate to be quantitatively evaluated. The GA then evaluates each candidate according to the fitness function. Only good individuals in the current population survive to the next generation while a bad one is eliminated from the selection process.

Reproduction selects individuals with high fitness values in the population, and through
crossover and mutation of such individuals, a new population is derived in which individuals may be even better fitted to their environment. The process of crossover involves two chromosomes swapping chunks of data. Mutation introduces slight changes into a small proportion of the population and is representative of an evolutionary step.

The rest of this paper is organized as follows: Section 2 describes multi-stage based genetic algorithm. Section 3 describes the proposed work. Section 4 consists of experiment and result analysis and section 5 consists of conclusion.

II. EXISTING METHOD

Multi-stage based genetic algorithm generates test cases for object-oriented programs [2]. MSGA includes two optimization stages. First it generates test cases, which are sequences of methods issue on an object of the class under test (CUT) and satisfy a given test criterion. Second it generates test data, which is a set of values for the arguments of the called methods.

In MSGA, two-point crossover and two-point mutation are used to generate test cases for next generation.

III. OBJECT-BASED EVOLUTIONARY ALGORITHM

To improve the performance of the GA it can either change its operators, parameters, data set and for various application domain. The new methodology has been implemented in OBEA that includes the rank selection, arithmetic crossover and two point random mutations have been attempted.

Structural Testing

An important technique used in testing of software is structural testing, in which a particular type of program element is selected for coverage [4]. Path coverage, statement coverage and branch coverage are an example of structural testing.

Statement coverage: In this the test case is executed in such a way that every statement of the code is executed at least once.

Branch coverage: A test method which aims to ensure that each possible branch from each decision point (e.g. “if” statement) is executed at least once, thus ensuring that all reachable code is executed.

Condition Coverage: It reports the true or false outcome of each condition. A condition is an operand of a logical operator that does not contain logical operators. Condition coverage measures the conditions independently of each other.

Path coverage: In this the test case is executed in such a way that every path is executed at least once.

In this paper branch coverage is considered as criteria to generate test cases.

Genetic Operators

Rank selection: In this type of selection, it first ranks the population and then every chromosome receives fitness from this ranking. The worst will have fitness 1, second worst 2 etc. and the best will have fitness N (number of chromosomes in population).

Arithmetic crossover: A crossover operator that linearly combines two parent chromosome vectors to produce two new offsprings according to the following equations:

\[
\text{Offspring}_1 = a \times \text{parent}_1 + (1-a) \times \text{parent}_2 \\
\text{Offspring}_2 = (1-a) \times \text{parent}_1 + a \times \text{parent}_1
\]

Where \(a\) is a random weighting factor.

Consider the following 2 parents (each consisting of 4 float genes) which have been selected for crossover:

- Parent 1: (0.3) (1.4) (0.2) (7.4)
- Parent 2: (0.5) (4.5) (0.1) (5.6)

If \(a = 0.7\), the following two offspring would be produced:

- Offspring 1: (0.36) (2.33) (0.17) (6.86)
- Offspring 2: (0.402) (2.981) (0.149) (6.842)

Two-point random mutation: The random positions for the mutations are determined. The positions of the mutation points are always varying to produce the offsprings.

Consider the following parent with five genes:

- Parent1: 3 4 5 6 5

Offspring: 3 2 5 6 8
**Fitness calculation:** The fitness value of the test case is calculated using two fitness functions $f_1$ and $f_2$ since we are considering two objectives for optimization namely maximization of coverage and minimization of execution time. The functions are given by

$$f_1(x) = \frac{\text{no of executed decision outcomes}}{\text{Total no of decision outcomes}}$$

$$f_2(x) = \frac{\text{execution time of a test case}}{\text{total execution time}}$$

$$f(x) = \max \{f_1(x)\} + \min \{f_2(x)\}$$

The optimal test cases are chosen by their fitness values that are calculated based on the values from these two fitness functions. The best test case will have value equal to 1 for the function $f_1$ and will have the value approximately equal to zero for the function $f_2$.

**IV. EXPERIMENTS AND RESULT ANALYSIS**

The object-based evolutionary algorithm is used to generate test cases for the object-oriented programs. The GA has been tried several times with different values of population size (50, 75, 100...), mutation probability and crossover probability (merely equals 1). The termination criteria occurs when maximum number of generations reached i.e., 1000. The effectiveness of the GA is studied by applying the algorithm on simple java classes.

The execution of genetic algorithm repeats till all the test requirements is satisfied or the maximum number of generations is reached or until getting the optimal solution with fitness value nearly equals 1. The final test cases are the minimal test set obtained from the resultant test cases of GA which satisfy all the target test requirements. Thus, the minimum no. of test cases is found for testing a given class.

The sample programs have the no. of conditions in the range 10 to 38 with Scheduling algorithm having 15 conditions and 8-queens problem containing 38 conditions. From Fig.1 and Fig. 2 it is inferred that when the number of conditions increases the execution time is also increased.

The results obtained for sample java programs are given below:

<table>
<thead>
<tr>
<th>Sample programs</th>
<th>No. of conditions</th>
<th>No. of test cases</th>
<th>Execution time (ms)</th>
<th>Branch coverage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Binary Search Tree</td>
<td>15</td>
<td>55</td>
<td>2355</td>
<td>94.3</td>
</tr>
<tr>
<td>Disjoint-Set</td>
<td>13</td>
<td>40</td>
<td>2092</td>
<td>95.6</td>
</tr>
<tr>
<td>Dequeue</td>
<td>10</td>
<td>30</td>
<td>1538</td>
<td>95</td>
</tr>
<tr>
<td>Quick sort</td>
<td>10</td>
<td>65</td>
<td>1963</td>
<td>94.2</td>
</tr>
<tr>
<td>Doubly Linked List</td>
<td>25</td>
<td>40</td>
<td>2940</td>
<td>92.5</td>
</tr>
<tr>
<td>OrdSet</td>
<td>38</td>
<td>35</td>
<td>3135</td>
<td>91</td>
</tr>
</tbody>
</table>

![Figure 1: Graph Showing Branch Coverage Obtained for GA](image1)

![Figure 2: Graph Showing Execution Time for GA](image2)

**V. CONCLUSION**

Thus, the object-based evolutionary algorithm has been used to generate test cases. The fitness depends on the branch coverage and execution time of the test cases. The results for sample java programs show that when the number of
conditions increases the execution time will be increased. By varying the crossover, mutation and selection techniques this genetic algorithm can be tried for the larger programs with higher complexity.

References


