Application of Genetic Algorithm and Tabu Search in Software Testing

Prashil R. Gupta, Abhishek Rathore, Atul Bohara, Lakshmi Prashanth T S, Praveen Ranjan Srivastava
Computer Science & Information System Department
Birla Institute of Technology and Science, Pilani
Rajasthan, India-333031
{ guptaprashil , rathore.abhishek87, atul.bohara, prash.130, praveensrivastava}@gmail.com

ABSTRACT
This paper presents a technique for automatic test-data generation in software testing. The proposed approach is based on genetic and tabu search algorithms. It combines the strength of two metaheuristic techniques and produces efficient results. The conventional approach for test-data generation using genetic algorithm is modified by applying a tabu search heuristic in mutation step. It also incorporates backtracking process to move search away from local optima. The experimental results show that the algorithm is effective in providing test data and its performance is better than simple genetic algorithm.

Categories and Subject Descriptors
H.4.m [Information Systems Applications]: Miscellaneous
D.2.5 [Software Engineering]: Testing and Debugging- testing tools

General Terms
 Algorithms, Measurement, Performance, Verification.

Keywords
Software Testing, Structural Testing, Test-data generation, Genetic Algorithm, Tabu Search, Control Dependence Graph.

1. INTRODUCTION
Software engineering is a discipline concerned with all aspects of software right from development to its retirement [1]. Software testing is a phase of software development life cycle aimed at detecting faults in software in order to ensure software quality. As software testing consumes about 50% of software development effort [2], test data generation plays an important role.

Various approaches for test data generation have been automated. These can be classified into three broad categories: random, static and dynamic techniques [3, 4]. Some of the dynamic methods of automatic test data generation using metaheuristic techniques treat testing problem as search space or optimization problem. Due to the complexity of testing process, these techniques find a large space to search. Some of the metaheuristic techniques face the problem of local optimum, when applied to software testing.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. COMPUTE’11, March 25-26, Bangalore, Karnataka, India Copyright 2011 ACM 978-1-4503-0750-5/11/03 ...$10.00.

Local optimum is a solution that is best within neighboring space but not globally. The search algorithms have a tendency to converge prematurely to local optimum. Because of this test data generated will not satisfy the test requirements. Particularly, Genetic algorithm suffers from problems like slow convergence, blind search and risk of getting stuck into local optimum solution [5].

This paper presents an automatic test-data generation technique based on genetic and tabu search algorithms. Genetic algorithm generates new test data from previously generated good candidates. It guides the direction of search by evaluating candidate test data using control-dependence graph (CDG) [6] of the program. The tabu search is added to the mutation step of genetic algorithm to reduce the randomness and execution time of search.

2. BACKGROUND
Several approaches have been proposed for automatic test data generation under all categories [7, 8]. Dynamic techniques avoid many problems faced by static techniques and are subject of recent studies. The application of metaheuristic techniques in the field of software engineering was proposed by SEMINAL network [4].

2.1 Genetic Algorithms
Genetic Algorithm (GA) is an optimization heuristic technique that is inspired by natural evolution and Darwin’s principal of survival of the fittest. It uses selection, recombination and mutation operators to generate new candidates (chromosomes) from existing population [5].

An approach for application of GA in structural testing was proposed in [9, 10]. [10] used fitness function based on predicates of the branch.

2.2 Tabu search
Tabu search (TS) is a memory based adaptive algorithm that guides local heuristic search procedure to explore solution space beyond local optimality [10, 11]. As the search proceeds, the content of memory keeps on changing. This memory is used to tabu paths which are already explored, so that the search will not be repeated in these paths.

Some papers like [5], have proposed hybrid of two heuristic techniques based on tabu and genetic algorithms for software testing. It used TS as mutation operator.

3. PROPOSED STRATEGY
In the proposed algorithm a combination of genetic and tabu based approach is used to optimally generate input test cases so that every branch of the program is covered.
Various notations used in the following sections are: POP is the size of population, MAXLIM is total number of allowed iteration for a TARGET NODE (TN). MAXC, currAvg and threshold are the variables used for backtracking of the algorithm. The explanation of all the variables used is provided in following steps:

### 3.1 Strategy for Numeric Input Data

**Step 1: Initial Population Generation**

Initially random test cases, denoted by POP are generated for the program under test through random selection [9, 6]. These inputs are called chromosomes.

A control flow graph (CFG) is constructed showing the graphical representation of the program, which is then converted to a control dependence graph (CDG) [6]. The nodes and edges in CDG represent statements and control dependencies among them respectively. To get the coverage data, the program under test is instrumented. Instrumentation of the program is obtained by inserting some statements in it without changing its logic.

Among the uncovered nodes, the node which has the minimum height in the CDG is considered as the TARGET NODE (TN). In case of more than one node with same minimum height, the TN is chosen randomly among these.

**Step 2: Fitness Value Calculation**

After the random population is generated the fitness value of each chromosome of the population is calculated based on TN. Fitness of a chromosome depends on the number of predicates common in its coverage path and predicates of the TN in CDG. The coverage path of a chromosome is the path in CDG it traverses when the program is executed with the chromosome as input.

A predicate is combination of a node and an outgoing edge from it. Predicates of TN are given by the acyclic path from root of CDG to this node. The test case which covers maximum number of predicates in common with the TN is given highest fitness value. Selection of the population of next generation is biased towards fitter test cases. If a test case has a number of predicates in common with that of TN then it satisfies majority of the conditions to reach the TN. So there is a better chance to get a test case which covers the TN by using these inputs.

Figure 1 shows a CDG for a particular program. Here E represents Entry node of the program and T and F stand for True and False edges respectively. As an illustration, if node 7 is taken as the TN then its predicates in CDG, are \{ET, IF, SF\}. If predicates for a test case are \{ET, IT, 2T\} then there is 1 predicate (ET) common in TN and input test case. So the fitness value of this test case with respect to the TN is 1.

**Step 3: Crossover and Mutation**

Next, some of the test cases are selected as parents of the next generation. This selection is biased towards fitter test cases. A fitter test case may be selected multiple times or a test case with low fitness value may be selected less number of times based on crossover probability (Cp) [5]. Some of these test cases are considered for mutation based on the mutation probability (Mp) [5]. Given a test case ti with fitness fti, Cp is calculated as \( Cp = \frac{fti}{\sum_{j} fti} \) (0<i<POP). A random number r (0<r<1) is generated based on which test case ti is selected. This process is continued until required number of parents is selected for the next generation. Recombination using one point crossover [6] and mutation is used to produce next generation.

**Step 4: Population of Tabu Lists**

Next, the program is executed with new set of test cases to find the fitness values of each test case. If the TN changes during this process then existing population is deleted and a new population is generated for the new TN, otherwise this process is repeated for MAXC times [3]. Short term tabu list is populated in this process which consists of the predicates that are frequently found in inputs having low fitness values. This is a frequency based tabu list [12]. This tabu list helps in preventing the generation of a test case with predicate values seen more often in bad test cases. Any such tabooed predicate is deleted from this short term tabu list after its tabu tenure is over. Tabu tenure is varied according to the need of the program, which lies in the interval \([nb, 5nb]\), where nb is the number of bits of the inputs of the program [3].

Long term tabu list is another list which is used in backtracking of the search procedure, as explained in step 5. Long term tabu list consists of those test cases which have the highest fitness value in each iteration of the algorithm, whose size (LSIZE) is decided initially. One of the contents of this list is replaced by a new test case if it has a better fitness value.

**Step 5: Backtracking**

Backtracking of search procedure can happen in two cases:

1. **Number of iterations (IT2) for a particular TN has exceeded MAXC as specified in step 3.** This signifies that the algorithm is in local optima. To move away from this, diversification [3] is employed by the use of backtracking.

2. **If the average fitness value (currAvg) of the population is less than threshold.** The threshold is half of the moving average of the fitness values of previous populations. Initial moving average is average of first and second population’s fitness value. After that, in each iteration the moving average is calculated as:

   \[
   \text{Moving average} = \frac{\text{old moving Avg} + \text{curr Avg}}{2}
   \]

During backtracking (POP/2) test cases are selected from long term tabu list and same number of test cases is generated randomly. Both of these act as parents for the next generation of
populations. The process of crossover takes place as specified in step 3. Advantage of this is that the search moves away from local optima using a test case which is in a long term tabu list with the introduction of randomness by crossover operation.

This process is repeated for a number of iterations, but if it exceeds MAXLIM, the TN is declared unreachable and new TN is selected. This continues until all the target nodes are processed i.e. covered or declared unreachable.

3.2 Strategy for String Input Data
The proposed strategy can also be applied on programs consisting of character string(s) as input. The real problem faced while applying automatic test-data generation strategies to this type of programs is that the fitness function (cost function) is not obvious [13]. Due to this reason, the above mentioned approach needs some modification for string inputs.

The goal here will be same as in case of numeric inputs-to traverse the TN. But in this case, rather than considering CDG paths as fitness criteria, a new fitness function is proposed [14]:

$$F = \sum_{i=0}^{L_1-1} str1[i] \cdot w^{L_1-i-1} - \sum_{i=0}^{L_2-1} str2[i] \cdot w^{L_2-i-1}$$

Here L1 and L2 are lengths of two strings str1 and str2. L is the maximum of L1 and L2, w is taken 128 (number of printable characters). The fitness function gives a numerical difference between the two strings.

The approach will be different for different types of string based programs, and so the approach is program specific. For a program consisting string predicates of type strcmp(str1, str2) op 0, where op can be one of the three <, =, >, this algorithm works in following steps:

**Step 1: Initial Population Generation**
This step works the same way as explained for numeric inputs.

**Step 2: Fitness Value Calculation**
The fitness of initial population is calculated by the fitness function given above.

**Step 3: Mutation**
A slight difference is introduced here. Basic mutation operation is applied on test case, rather than crossover and tabu based mutation.

**Step 4: String input processing**
1. Determine direction (dir) using instrumentation of code. The value of dir can be any of the three >, < or =. dir is recorded at each conditional statement of the program under test consisting string predicate.
2. Move str1 in dir of str2 with AMOUNT. The initial value of AMOUNT is 1.
3. Change AMOUNT as per the move, if moving is beneficial, AMOUNT is doubled otherwise set AMOUNT to initial value 1. This step is adaptive and aimed at reducing number of total iterations needed to cover the TN.

4. ANALYSIS OF THE STRATEGY
Important novelties of the approach include use of TS with basic GA and use of CDG to guide the search. TS algorithm avoids the local optima problem of GA. Whenever the search gets stuck in a local optimum, backtracking process is used. The result of backtracking is a state having characteristics of current best solution as well as random characteristics.

CDG is used to guide the search and measure its success as it is proved to be effective [6]. Since CDG is based on the control dependencies of the program, it helps in determining TN. Studies have shown that CDG can scale for large programs. A linear relationship exists between the size of CDG and number of executable statements of the program [15]. Some techniques like inserting region nodes are used for structuring CDG hierarchically and representing loops and predicate statements efficiently [16].

As the approach is based on two metaheuristic techniques, the number of iterations taken to generate test-data for a program under test may be inconsistent. Also the basic algorithm needs to be modified for input types other than numeric. For character string inputs, the approach has to be changed as explained in the previous section.

5. CASE STUDY
To determine the effectiveness of approach, the algorithm was implemented in C and tested for Triangle Classifier Program. This program takes 3 sides of triangle as input, and classifies the triangle as equilateral, isosceles, scalene or no triangle. It gives invalid range error message if given input in not in appropriate range. The CDG for this program is shown in figure 2.

![Figure 2: CDG for Triangle Classifier Program](image)

**Table 1**

<table>
<thead>
<tr>
<th>Input Range</th>
<th>Tests</th>
<th>Nodes Uncovered</th>
<th>Tests</th>
<th>Nodes Uncovered</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-5</td>
<td>10</td>
<td>None</td>
<td>290</td>
<td>None</td>
</tr>
<tr>
<td>0-10</td>
<td>65</td>
<td>None</td>
<td>295</td>
<td>None</td>
</tr>
<tr>
<td>0-15</td>
<td>75</td>
<td>None</td>
<td>605</td>
<td>None</td>
</tr>
<tr>
<td>0-50</td>
<td>700</td>
<td>None</td>
<td>2860</td>
<td>None</td>
</tr>
<tr>
<td>0-100</td>
<td>765</td>
<td>None</td>
<td>65535</td>
<td>Node 8</td>
</tr>
</tbody>
</table>
6. COMPARISION WITH EXISTING WORK

Genetic based approach of test case generation is used in paper [6]. Here a random set of test cases are generated to form initial population. Fitness value calculation is based on current target node in CDG. As the current target node keeps changing the fitness value calculation is dynamic.

Tabu based approach [3] can be used for test case generation using two memory lists, a short term tabu list which is used for intensification and a long term tabu list, used for diversification. A hybrid approach in paper [5] combines some features of genetic and tabu for optimal test case generation.

Proposed algorithm uses the genetic based approach to gradually intensify the search procedure. Using the CDG for evaluation of fitness makes the algorithm more efficient. The mutation is performed efficiently as specified in paper [5] based on hybrid approach. Risk of getting stuck in local optima is eliminated by the use of backtracking which is guided by a long term tabu list consisting of good solutions. The results in previous section affirm that this algorithm is efficient. Statistical data about comparison between proposed algorithm and genetic based algorithm are plotted in figure 3:

![Figure 3: Comparison of Results](image)

7. CONCLUSION

This paper presents strategy for automatic test-data generation using a combination of genetic and tabu search algorithms to obtain branch coverage criterion.

The results of case study and statistical data support the claim that this algorithm performs better than other related strategies. The strategy for string input data requires further exploration. We are planning to enhance this strategy in future work.

8. REFERENCES