State Based Problem Using Evolutionary Technique: Test Data Generation for Object Oriented Systems in presence

Vipin Kumar Sharma[1], Harish Pratap Singh[2], Manoj Kumar[3]

[1]*Head of Department(Assistant Professor) , Department of Computer Science, Krishna College of Science & IT, Bijnor, Uttar Pradesh, India*
[1]*Assistant Professor, Department of Computer Science, Krishna College of Science & IT, Bijnor, Uttar Pradesh, India*
[3]*Research Scholar, Department of Computer Science, Bhagwant University, Ajmer, Rajasthan, India*

Abstract

Software systems have become an integral part of all engineering systems and are providing assistance in a variety of engineering activities. However, the quality of assistance provided depends not only on the coded functionality, but also on the fact that the software is free from defects. Since formally proving a software system to be correct becomes more and more difficult as the size and complexity increases, one of the pragmatic ways of ensuring software correctness is through testing. Test Data Generation Approaches for Object Oriented Systems, in presence of State Based Problem, using evolutionary techniques. During testing the software to be tested is executed on a test set of test cases, selected from the input domain, and the results are evaluated.

Keywords— *Test Data Generation, Unit Testing, object-oriented software, evolutionary techniques, Genetic Algorithms*

1. INTRODUCTION

Software testing may defined as the process of executing a software system to determine wether the system work according to our need and provide desired output. Unlike static verification and validation approaches where software source code is the object of analysis, testing requires an executable. Testing is important in development of any software. The quality of any software product is checked and ensured through the testing. Testing object oriented-software is slightly different than the traditional once. The data is object oriented is wrapped with behaviour. The generation of test cases is a time consuming and error prone process because it is done by manually. To automate this process, search-based software testing offers a possible solution, which is based on meta-heuristic search techniques, evolutionary algorithm etc.

The focus of this paper is test data generation approaches for object-oriented software using evolutionary algorithms. To reach to a research proposal in test data generation approaches for object-oriented software, we begin, in section 2- give brief introduction about Unit Testing of Conventional Procedure-Oriented Software This is followed by an introduction to the object-oriented software testing, , in section 3.In section 4, explain the different search techniques used in test data generation for object oriented systems. In section 5, give the introduction about genetic algorithm so that we can use in object oriented system. Section 6 and 7, motivation and research objective is describes. Finally, the conclusion is outlined in last section.

2. UNIT TESTING OF CONVENTIONAL PROCEDURE-ORIENTED SOFTWARE

Much of the research in testing conventional, procedure-oriented, systems has focused on test case selection for testing individual modules, i.e., single procedures or functions, at the unit level (Zhu 1997). Furthermore, ever since Goodenough and Gerhart (Goodenough 1975) pointed out that the limitations of testing - testing can never be s white box criteria, i.e., control flow (Zhu 1997) and data flow criteria (Rapps 1985, Frankl 1988). The definitions of these adequacy criteria are based on the flow graph of the program structure - a directed graph in which nodes represent linear sequences of statements and edges represent transfer of control. Control flow criteria use the control flow graph directly to specify test requirements

whereas data flow criteria use a data flow information augmented control flow graph. The actual description of the data flow testing methods is based on the investigation of the ways in which values are associated with respective variables and how these associations can influence the execution of the program. Another class of criteria is the fault-based criteria. These criteria measure the quality of a test set according to its ability to detect specific faults. This idea of generating test cases to detect specific faults led to the definition of mutation analysis and mutation testing (DeMillo 1978).

3. OBJECT-ORIENTED SOFTWARE TESTING

For large object-oriented software also, testing begins with unit testing and ends up with acceptance testing. However, the encapsulation of data and functions, as it exists in objects, results in a different interpretation of the first two levels of testing which is as follows:

(a) Object-Oriented unit testing is testing (i) a single member function of a class or a non-member function, and (ii) a single class of objects.

(b) Object-Oriented integration testing, involves: (i) testing behavioral dependencies or interactions in collaborations of class instances, which is also referred to as cluster testing, and (ii) subsystem integration testing.

This reinterpretation brings into focus a very fundamental question: "Is the testing of object-oriented software different from testing non-object-oriented software?" Although there exists a general agreement that it is different (Binder 1995), researchers have tried, with a fair degree of success, to adapt conventional software testing techniques to testing object-oriented software (Zweben 1992, Parrish 1993, Offutt 1995).

4. SEARCH TECHNIQUES USED IN TEST DATA GENERATION

In this section, we review the search techniques for the generation of test data.

A.Meta-heuristic Search Techniques

Meta-heuristic search methods are top level structure which uses heuristics to find solutions to problems without the need to perform a full exhaustive enumeration of a search space. They are therefore well suited for combinatorial problems for finding good solutions of fair computational cost. These type of problem have been classified under NP- complete and NP-hard problem or algorithm based on polynomial time are in existence but not have any practical implementation are done.. These frameworks are not standalone algorithms in their own right, but rather "strategies" that are ready for adaptation to particular problems. A good encoding will ensure that candidate solutions sharing a number of similar properties will be "neighbors" in encoded solution space. Another key decision is the definition of a problem-specific objective function, which the search uses as a guide to the quality of candidate solutions.

B. Hill climbing

Hill climbing is a popular technique for local search . In hill climbing, initial solution is randomly chosen from the search state space as a initial point for start searching. The neighbors' of this initial solution is explored. It replaces the current solution, if a better solution is found. The neighbors' of the new solution is then explored. Again current solution is replaced if the better solution is found and so on, till no other better solution neighbor found for the current solution.

In a "steepest ascent" climbing strategy, all neighbors' are evaluated, with the neighbor offering the greatest improvement chosen to replace the current solution. In a "random ascent" strategy neighbors are examined at random and the first neighbor to provide an improvement get selected.

Hill climbing is simple and gives fast conclusions. It is easy for Hill climbing search to obtain sub optimal solution when it leads to local optimal solution but global optimal solution. In such problems, at the peak of a hill the search becomes trapped, not able to explore other point of area of the state search space. The exploring will stuck along plateau in the landscape. In such circumstances, no neighboring solution is deemed to provide an improvement over the current solution, since they all have a same point of value. The highly dependent results found in the hill climbing in non-trivial landscapes.

A common extension to this algorithm is to incorporate with to restart again with different starting solutions, to sample more of the search space and minimize this problem as much as possible.

C. Simulated Annealing

Simulated annealing originates from the technique of chemical process of annealing. If the material of solid type heated at its melting point and again cooled down and gets back into solid state then the structural based properties of cooled solid material depend on the rate of cooling.

It is important to have a less dependent search framework on the starting solution. The working principle of simulated

annealing is same as hill climbing. Simulated annealing restricts the movement around the search space and accept probabilistically poorer. The probability of selecting p of a minor solution changes as the search get advance, and is calculated as:

$$p = e^{-\delta}t$$

Where δ is the difference in objective value between the current solution and the neighboring solution being considered, and temperature t is a control parameter. According to a cooling schedule the temperature gets cooled. In starting to remove the dependency on the starting solution and to allow free movement in search space the temperature should be high at initial stage. As we continue the search, the temperature gets decreased. If cooling is fast then not enough search space is investigated and chances of search get increased to stuck in local optima.

D. Evolutionary Algorithms

Evolutionary algorithm is a search strategy based on the simulated evolution to evaluate the candidate solution by using operators of genetics and natural selection.Genetic algorithm is the most popular method of evolutionary algorithm given by John Holland late sixties. Genetic algorithm based on the natural genetic evolution strategies.

In genetic algorithm, the search is based on the recombination mechanism between individuals to create new individuals and uses mutation for some modification in the solutions. This recombination mechanism and mutation method was developed independently and now in recent work both ideas gets combine to create a better solution.

5. GENETIC ALGORITHM

Genetic algorithm inspired from natural or biological evolution. Genetic algorithm based on the encoding of candidate solutions and genetic chromosome. Solutions are also known as individuals or chromosomes. They are called search problems used to solve optimization problem.

The part of the solution is known as genes. Position of gene in chromosome has specified location called locus. In genetic algorithm the encoded structure for the solution is called the genotype and the decoded structure known as the phenotype. Many applications genetic algorithm the genotype is used as a string of binary digits. In a population each Individuals compete for resources. Those individuals which get success in each competition will generate more offspring than those individuals that not perform well. Those individuals having high fitness function will become more for the environment. The population performs crossover and mutation to produce optimized solution and successive populations known as generations.

The crossover operator produce two offspring by taking two parent from the population based on fitness value. In single crossover, two new is produce by chossing random point for recomibination. A recombination of two individuals 1101100100110110 and 1101111000011110in encoded form, with a single-point crossover chosen to take place at locus

1011	00100110110	101111000011110
1011	11000011110	101100100110110

Crossover produces two offspring. Multiple point crossover operators perform a recombination by choosing a fixed position.

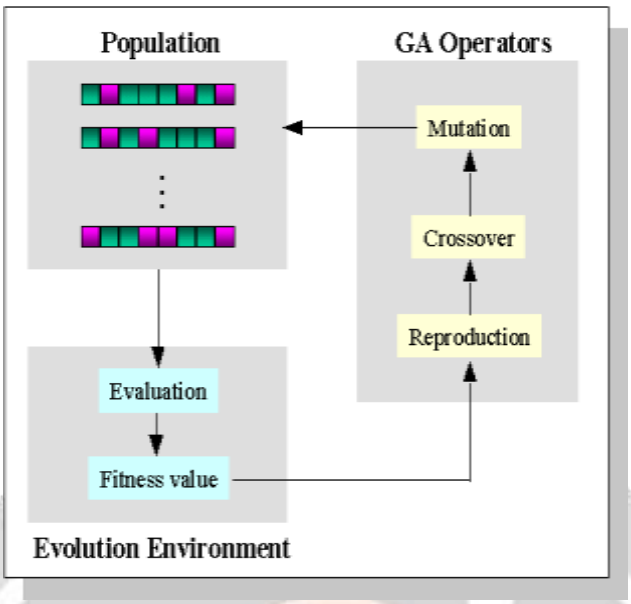

Fig1. Genetic Algorithm Cycle

There are various mechanism for the selection of individuals used to create offspring for the next generation. The main focus on selection is the fitness function the individuals. Fitness function of an individual is the value obtained from the objective function or the scaled value. The idea of selection is to favor the fitter individuals. If the strategy is too weak then it will result in more exploration

Roulette wheel selection is a common selection method in genetic algorithm .In this method each individual is allocated a slice of the wheel based on the fitness function. The wheel is then spun N times in order to pick N parents. At the end of each spin, the position of the wheel marker denotes an individual selected to be a parent for the next generation.

Linear ranking of individuals is a technique which proposes to circumvent this problem. Individuals are sorted by fitness, with selection done by providing rank, instead of direct use of fitness values. A linear ranking mechanism with bias Z, where $1 < Z \leq 2$, allocates a selective bias of Z to the top individual, a bias of 1.0 to the median candidate, and $2 - Z$ to the bottom candidate. Throughout the search constant bias is applied, selective pressure is more constant and controlled [Whi89].

Tournament selection is a noisy but fast rank selection algorithm. The population is not arranging according to fitness function. Two individuals are chosen at random from the population. A random number, $0 < r \leq 1$, is then chosen. If r is less than p (where p is the probability of the better individual get selected), the two individuals having highest fitness function wins and chosen to be a parent, otherwise the less fit individual is chosen. The competing individuals are returned to the population for further possible selection. This is repeated N times until the required number of parents have been selected. In all probability, every individual is sampled twice, with the best individual selected for reproduction twice, the median individual once, with the worst individual remaining unselected. The resulting selective bias is dependent on p. If $p = 1$, then in all probability a ranking with a bias of 2.0 towards the best individual is produced. If $0.5 < p \leq 1$, then the bias is less than 2.0.

Once the set of parents has been selected, recombination can take place to form the next generation. Crossover is applied to individuals selected at random with a probability p_c (referred to as the crossover rate or crossover probability). If crossover takes place, the offspring are inserted into the new population. If crossover does not take place, the parents are simply copied into the new population. After recombination, a stage of mutation is employed, which is responsible for introducing or reintroducing genetic material into the search, in the interests of maintaining diversification. This is usually achieved by flipping bits of the binary strings at some low probability rate p_m, which is usually less than 0.01.

6. MOTIVATION OF THE PROBLEM

Meta-heuristic search technique used for the generations of test data increases the interest of researcher in recent years. Today, the search techniques apply to test data generation has mainly focused on generating inputs based on input-output behavior for test objects. The main of work is to extend the method with state behavior for test objects. This work has several challenges because test goals require input sequences based on state test objects. Next problem is the use of

internal variable in generating test data such as counters, flags and enumerations. These variables are in charge of organizing the state of the test object. But the use of these internal variables leads to loss of information with respect to original input conditions that fulfill the certain goals. Due to this search receive less information and get failed in finding test data.

The work proposes an evolutionary test data generation approach that allows generating input sequences and dealing with the internal variables through chaining approach and hybridization method. The main idea of chaining approach to find input sequence with the involvement of input variables which need to executed for test goals. These input sequences are executed and used in past unavailable information for search and guide the promising and unexplored areas of test objects. The experiment performs show the value for the test objects in state and input output behavior. Branch coverage are obtained for all test objects.

In test objects present two major challenges for evolutionary structural test data generation:

A. Input Sequences

The standard evolutionary approach generates input vectors for single function calls. Test objects with states may require a sequence of calls to be generated in order for certain structures to be covered. This sequence may include calls to several different functions. Take the example of the C module representing a stack. In order to cover the statements that remove an element from the top of the stack - nodes b and c in the pop function - the push function needs to have first been called to put an element onto the stack, because initially, the stack is empty.

The state of the stack is managed by the elements array These state variables are declared using the static C keyword, which hides them from external calling processes. Therefore, the state of the stack can only be changed by invoking its visible functions. When the stack is empty, no calls to pop alone will lead to nodes b and c being executed. A state variable of a test object is an internal variable whose value is retained after the termination of a function of the test object until a function of the test object is next called.

B. Internal Variable Problem

The internal variables used in the conditions of programs can result in a degree of "information loss" when computing branch distance values, producing coarse or flat objective function landscapes for target structures within the program. This will turn the results for receiving less instruction about the search, making it less likely - if not impossible - that the required test data will be found. The degree of "difficulty" for the search depends on the level of information lost, which in turn depends on the type of the internal variable. Some internal variables only result in a small amount of information loss, which may not affect the success of the search.

Test data is generated for atomic function calls. However functions and part or element of top system levels can store internal data, and can exhibit different behaviors based on the state of that data. This presents new challenges to the test data generation method. The first challenge is to generate a sequence of inputs to the test object, since certain program structures may require the test object to be in a particular state in order for them to be covered. For example, statements popping a value from a stack would not normally be covered unless the stack was in a non-empty state. The second challenge involves the problem of internal variables. State-based test objects by their very nature contain internal variables in order to manage their state. This can become problematic when internal variables like flags are used to manage or query the state, because the search may have difficulties in finding input sequences in order to cover certain structures within the program. In this work for internal variable problem various solutions obtained. Chaining approach is one possible solution for internal variable problem. The main idea of the chaining approach is to identify a sequence of statements that need to be executed prior to the target structure. These statements involve assignments to internal variables. They are executed, for information previously unavailable to the search, possibly guiding for unexplored areas of the test object's input domain. In this way, the chances of finding input data to troublesome structural targets may be improved.

7. CONCLUSION

In this paper, we discuss about the different techniques for the generation of test data. Test data for testing object oriented software includes test program which form and change the objects in order to achieve a certain aim. The work in the field of search-based structural test data generation has largely focused on the testing of individual program functions with input-output behavior. The approach described in this paper facilitates the automatic generation of test data for object oriented system using genetic algorithms.

REFERENCES

[1]. Chen, H.Y., Deng, Y.T., and Tse, T.H., 1997b, "ROCS: an object-oriented software design system at the class level based on the relevant observable technique," Tech. Rep. TR-97-08, Dept. of Computer Science, University of Hong Kong, Hong Kong, China.

[2]. [2] Chen, H.Y., Tse, T.H., Chan, F.T., and Chen, T.Y., 1998, "In black and white: an integrated approach to class-level testing of object-oriented programs," ACM Transactions on Softw. Engg. and Methodology, Vol.7, No.3, pp.250-295.

[3]. DeMillo, R.A., Lipton, R.J., and Sayward, F.G., 1978, "Hints on test data selection: help for the practicing programmer," IEEE Computer, Vol.11, (Apr.), pp.34-41.

[4]. Doong, R.K., and Frankl, P.G., 1994, "The ASTOOT approach to testing object-oriented programs," ACM Transactions on Softw. Engg. and Methodology, Vol.3, No.2, pp.101-130.

[5]. Frankl, P.G., and Weiss, S.N., 1993, "An experimental comparison of the effectiveness of branch testing and data flow testing," IEEE Trans. on Software Engg., Vol. 19, No.8 (Aug.), pp.774-787.

[6]. Goodenough, J.B., and Gerhart, S.L., 1975, "Towards a theory of test data selection," IEEE Trans. on Software. Engineering, Vol.1, No.2 (June), pp.156-162.

[7]. Gursaran, Porwal, R., and Caprihan, R., "Towards Test Data Generation for Clustering Testing using UML Sequence Diagrams and Genetic Algorithms," in Proceedings of the Sixth International Conference on Information Technology, CIT-03, Bhubaneswar, India, Dec 22-25, 2003, pp. 61-66.

[8]. Harrold, M.J., McGregor, J.D., and Fitzpatrick, K.J., 1992, "Incremental testing of object-oriented class structures," Proceedings of the 14th International Conference on Software Engineering, pp.68-79.

[9]. Jalote, P., 1989, "Testing the completeness of specifications," IEEE Trans. Software Engg., Vol. 15, No. 3, pp.526-531.

[10]. Jones, B. F., Eyres, D.E., and Sthamer, H.H., "A strategy for using genetic algorithms to automate branch and fault testing," The Computer Journal, vol. 41, no. 2, pp. 98-107, 1998.

[11]. Khor S., and Grogono, P., "Using a genetic algorithm and formal concept analysis to generate branch coverage test data automatically," in Proceedings of 19th International Conference on Automated Software Engineering, Linz, Austria, Sep. 20-24, 2004.

[12]. Korel, B., "Automated software test data generation," IEEE Transactions on Software Engineering, vol. 16, no. 8, pp. 870-879, Aug. 1990.

[13]. [13] Kung, D.C., Gao, J., Hsia, P., Toyoshima, Y., Chen, C., Kim, Y., and Song, Y., 1995, "Developing an object-oriented testing and maintenance environment," Commun. of ACM, Vol. 38, No.10 (Oct.), pp.75-86.

[14]. [14] Michael, C.C., McGraw, G. E., and Schatz, M. A., "Generating test data by evolution", IEEE Transactions on Software Engineering, vol. 27, no. 12, pp. 1085-1110, 2001.

[15]. [15] Overbeck, J., 1993, "Testing object-oriented software: State of the art and research directions," Proceedings 1st European International Conference on Software Testing, Analysis and Review, (Oct.), Jacksonville, Fla.

[16]. [16] Rapps, S., and Weyuker, E.J., 1985, "Selecting software test data using data flow information," IEEE Trans. Software Engg., Vol. 11, No. 4 (April), pp.367-375.

[17]. [17] S. Wappler, F. Lammermann, Using Evolutionary Algorithms for the Unit Testing of Object-Oriented Software, In GECCO '05: Proceedings of the 2005 Conference on Genetic and Evolutionary Computation, pages 1053-1060, Washington, D.C., USA, ACM Press, 2005

[18]. [18] S. Wappler, J. Wegener, Evolutionary Unit Testing of Object-Oriented Software Using a Hybrid Evolutionary Algorithm, In Proceedings of the IEEE World Congress on Computational Intelligence (WCCI-2006), pages 3227-3233, Vancouver, BC, Canada, IEEE Press, 2006

[19]. [19] Schach, S.R., 2014, "Testing: principles and practice," ACM Computing Surveys, Vol. 28, No.1 (March), pp.277-279.

[20]. [20] Whitley, D., "A genetic algorithm tutorial," Journal of Statistics and Computing, vol. 4, pp. 65-85, 2014.