Soft Computing Based Approaches for Software Testing: a Survey

Bipin Pandey, Rituraj Jain

Abstract- Software testing is the process of validation and verification of the software product which in turn deliver the reliable and quality oriented software product to users with lower maintenance cost, and more accurate and reliable results. Software testing effectiveness always depends on issues like generated test cases, prioritization of test cases etc. These issues demands on effort, time and cost of the testing. Many academicians and researchers are using soft computing based approached for better accuracy in testing. The aim of this research paper is to evaluate and compare soft computing approaches to do software testing and determine their usability and effectiveness.

Index Terms: Black Box Testing, Fuzzy Logic, Genetic Algorithms, Neural Network, Soft Computing, Software Testing, Tabu Search, White Box testing

I. INTRODUCTION

For validating and verifying software system under development, testing is the most important activity of software development life cycle. It is the method to measure how much software system is conforming to its requirements specification specified by users and to demonstrate its correct operation. However, software testing is a time consuming and expensive task and almost 50% of the software system development resources are consume for software testing [1], [2]. Testing techniques are classified as functional (black box) and structural (white box) testing. Functional testing is based on functional requirements whereas structural testing is done on code itself. Testing tools are available to do the testing either manually or automatically. It is found that automated software testing is better than manual testing.

Random Testing is the commonly used search heuristics in the industry for test case identification and design. Evolutionary Testing, which are based on evolutionary algorithms [3, 4] are now popular in the industry, is an automatic test case generation technique based on the application of evolution strategies [5], genetic algorithms [6, 7], genetic programming [8], or simulated annealing [9]. In this research paper, a survey of different soft computing approaches used for software testing techniques is presented.

II. SOFT COMPUTING APPROACHES

Soft computing refers to a collection of computational techniques which study, model, and analyze very complex phenomena: those for which more conventional methods have not yielded low cost, analytic, and complete solutions [10]. The principal constituents of Soft Computing (SC) are Fuzzy Logic (FL), Neural Computing (NC), Evolutionary Computation (EC), and Machine Learning (ML) which are based on the information processing in biological systems. Recognition of surrounding, making prediction, and planning are the basic tasks that can be performed by using the complex biological information processing system [11].

A. Genetic Algorithm

Genetic Algorithms (GAs) are global optimization methods. They are particularly useful for problems that involve searching parameter spaces in which there are many local minima. GAs operates on a population of potential solutions applying the principle of survival of the fittest to produce better and better approximations to a solution. GAs, differing from conventional search techniques, start with an initial set of random solutions called population. Each individual in the population is called a chromosome, representing a solution to the problem at hand. A chromosome is usually, but not necessarily, a binary bit string. The chromosomes evolve through successive iterations, called generations. During each generation, the chromosomes are evaluated, using some measure of fitness. To create the next generation, new chromosomes, which called offspring, are formed by merging two chromosomes from current generation using a crossover operator or modifying a chromosome using a mutation operator. A new generation is formed by selecting, according to the fitness values, some of the parents and offspring and rejecting others so as to keep the population size constant. After several generations, the algorithm converges to the best chromosome, which hopefully represents the optimum or sub-optimal solution to the problem [12] [13] [14] [15].

B. Fuzzy Logic

Fuzzy Logic (FL) processing the data by allowing partial set membership rather than crisp set membership or non-membership. Fuzzy Expert System consists of fuzzification unit that converts crisp values into fuzzified input [16]. It consists of inference engine that contains if then else rules and a defuzzification unit to convert the result in a readable form. FL incorporates a simple, rule-based IF X AND Y THEN Z approach to a solving problem rather than attempting to model a system mathematically [17].

C. Neural Computing

A neural network, more properly referred to as an 'artificial' neural network (ANN), is a computing system made up of a number of simple, highly interconnected processing elements, which process information by their dynamic state response to external inputs [18]. The representation of knowledge is distributed over these connections and "learning" is performed by changing certain values associated with such connections, not by programming [19].

Manuscript received May 2014

Bipin Pandey, Department of Computer Sc. & Engg., Vyas Institute of Engineering & Technology, Jodhpur, India.

Rituraj Jain, Department of Computer Sc. & Engg., Vyas Institute of Engineering & Technology, Jodhpur, India.

III. SOFT COMPUTING IN WHITE BOX TESTING

White-box testing or structural testing is a verification technique to examine if code written for fulfilling the required task works as expected or not. It can be done in the form of data flow testing or path testing. Path testing is an approach to ensure that every path through a program has been executed at least once and finding the set of test cases that will execute every path in this set of program path [20] [21]. In data flow testing program's control flow is under tracking in order to explore sequences of events related to status of data objects caused by creation, usage, modification or destruction with the intention of identifying any data anomalies [22]. So in data flow testing, the focus is on the points at which variables receive values and the points at which these values are used [23].

P.R. Srivastava and Tai [3] have developing variable length Genetic Algorithms that optimizing software testing efficiency and select the software path clusters which are weighted in accordance with the criticality of the path. Their approach uses a weighted CFG. Weights are assigned to the edges of the CFG by applying 80-20 rule i.e. more weights are assigned to critical edges. So 80 percentage of weight of incoming credit is given to loops and branches and the remaining 20 percentage of incoming credit is given to the edges in sequential path. The selection of parents for reproduction is done according to a probability distribution based on the individual's fitness values. More weight is assigned to a path which is more "critical". The crossover technique used is one point crossover done at the midpoint of the input bit string. Mutation is performed on a bit-by-bit basis. To perform mutation, for each chromosome in the offspring and for each bit within the chromosome, generate a random real number r in the range [0, 1]; if r < pm then mutate the bit. The summation of weights along the edges comprising a path determines criticality of path. Higher the summation more critical is path and therefore must be tested before other paths. In this way by identifying most critical paths that must be tested first, testing efficiency is increased.

Another test generation approach proposed by P.R Srivastava is based on path coverage testing [24]. The test data is generated for Resource Request algorithm using Ant Colony Optimization algorithm (ACO) and GA. *Resource request algorithm* is deadlock avoidance algorithm used for resource allocation by operating system to the processes in execution cycle [25]. The ACO algorithm is inspired from behavior of real ants where ants find closest possible route to a food source or destination. The ants generate chemical substance called pheromones which helps ants to follow the path. The pheromone content increases as more ants follow the trail. The possible paths of CFG are generated having maximum number of nodes. Using ACO, optimized path ensuring safety sequence in resource request algorithm is generated covering all edges of CFG. Using GA, suitable test data set is generated which covers the need for each process. The backbone of genetic process is the fitness function which counts number of times a particular data enters and continues the resource request algorithm. Higher the value of count, higher is chances of avoiding a deadlock. The test data with higher values of count is taken and genetic crossover and mutation is applied to yield better results. Simultaneously, poor test data is removed each time.

Girgis [26] has proposed a structural oriented automatic test data generation technique that uses a GA guided by the data flow dependencies in the program to search for test data to cover its def-use associations. A Control Flow Graph (CFG) is prepared for the program where each node represents a block in a program and the edges depict the control flow of the statements. Data flow analysis focuses on the interactions between variable definitions (defs) and references (uses) in a program. In his approach, GA accepts instrumented version of the program under test, the list of def-use sets to be covered, the number of input variables, and the domain and the precision of each input variables as an input. A binary vector is used to represent a chromosome. The length of the input is determined by the domain and the precision. Each chromosome represents a test case for a program which is represented by a binary string of specified length. Each chromosome (as a test case) is represented by a binary string of length m. Initial population with m-bit strings pop_size is randomly generated where pop_size is the population size. Each chromosome is converted to k decimal numbers representing values of k input variables x_1, …, x_k (i.e. a test case). Selection is done by roulette wheel selection and proposed random selection method. The effective test cases then become parents of the new population. If none is effective then all the individuals are chosen as the parents. In the recombination phase, we use two operators, crossover and mutation. Results of the his approach are more effective as compared to the random testing technique. The proposed selection method generates better results than the roulette wheel selection method.

Yeresime Suresh et. al [27] proposed an approach using genetic algorithm for generating test data automatically which is reducing the test effort and time of a tester. The test data is referred to as population in GA. In initial population, each individual bit string (chromosome) is a test data. This set of chromosomes is used to generate test data for feasible basis paths. The system for generating automated test data for feasible basis paths using GA first randomly generates the initial population, evaluates the individual chromosome based on the fitness function value and applies the GA operations such as elitism selection, two point crossover operation and bit wise mutation to produce next generation. This iterative process stops when the GA finds optimal test data. After the generation of initial test data randomly, GA was iterated for 500 generations as in practicality computation time should be finite. This paper makes use of a fitness function based on the condition of the predicate node. The results of Yeresime Suresh et. al research are an indication that GA is more effective and efficient in generating automated test data rather than random testing.

Premal B. Nirpal et al. [28] proposed genetic algorithms based approach that can automatically generate test cases to test selected path. This algorithm takes a selected path as a target and executes sequences of operators iteratively for test cases to evolve. The evolved test case can lead the program execution to achieve the target path. To generate path-oriented test data for the program under test using GA, there are five steps wiz, Control flow graph construction, Target path selection, Fitness function construction, Program instrumentation, Test data generation and the instrumented program execution.

The quality of test cases produces by genetic algorithms is higher than the quality of test cases produced by random way because the algorithm can direct the generation of test cases to the desirable range fast. The of their research shows that genetic algorithms are useful in reducing the time required for lengthy testing meaningfully by generating test cases for path testing.

Jasmine Minj et al. [29] propose a technique to generate test cases from UML State diagram that is based on path oriented approach. Genetic algorithm is used with stack based approach to get the optimized feasible test cases. Effectiveness of test cases generated from UML Statechart is measured by state coverage, transition coverage and transition pair coverage criteria. They present path-oriented test data generation which aims to generate feasible test cases that covers every possible path in the program. UML Statechart is converted into intermediate graph. Then the predicates found in the intermediate graph are represented in the form of binary bits which is taken as chromosome. Based on predicates, traverse the graph using DFS for test sequence generation. Cost of each path is calculated using McCabe's formula of cyclomatic complexity formula. Fitness function is calculated by the cost of path and stack weight for each path. Selection is done using roulette wheel method and the individual probability is calculated based on the fitness of the individual. Then crossover operation will be applied and recombines the selected pairs of individuals. Bits are mutated which helps in introducing diversity into the genetic pool. It adds new individuals randomly to the population and thereby avoids solution being struck in the local optima. Their results show that generated test cases using these methods are effective, efficient and optimized.

Eugenia Díaz et al. [30] presented a tabu search metaheuristic algorithm for the automatic generation of structural software tests. The developed test generator has a cost function for intensifying the search and another for diversifying the search that is used when the intensification is not successful. It also combines the use of memory with a backtracking process to avoid getting stuck in local minima. The proposed method (TSGen) has the goal of covering all the branches of the program. Their method generates tests (partial solutions) based on the test that is the Current Solution (CS) and executes them as input for the program. Using the Current Solution, TSGen generates a set of neighboring test candidates and checks whether it is a tabu test. A test is tabu if it is stored in the TSGen memory. If a generated test is not tabu, the instrumented program under test is executed to check which branches (nodes) it has covered and the cost incurred by said test. However, if a generated test is tabu, it will be rejected. During the search process, the best solutions found are stored together with their costs in the CFG. Thus, when an executed test has a lower cost in a CFG node than the current cost stored in that node, that test is stored as the best solution for that node.

IV. SOFT COMPUTING IN BLACK BOX TESTING

Black box testing (also called functional testing) is testing that focuses solely on the outputs generated in response to selected inputs and execution conditions and ignores the internal mechanism of a system or component and. With black box testing, the software tester does not (or should not) have access to the source code itself.

Francisca Eanuelle et al. [31] presented GA based technique to generate good test plans for functionality testing. The test plan or test sequence totally relies on the experts or the people who understand the application well. The emphasis is given on the fact that an error in a program may propagate from the previous operations executed instead of the last operation. They have chosen the operation of large granularity so that the sequence of operation that leads application to inconsistent state can be identified. The objective is to find the sequence of operations which leads the system in an inconsistent state. Fitness value is calculated and larger the value of fitness function, better the sequence is considered which is likely to take the application to an inconsistent state. The results have shown that the GA improves the quality of the test plans. Their technique generates good test plans in an unbiased way but this requires computer applications to be tested more thoroughly. The approach does not use the structure of the application or the program flow.

Mark Last et al. [32] proposed a new Fuzzy-Based Age Extension of Genetic Algorithms (FAexGA) to generation of effective test. The basic idea is to eliminate "bad" test cases that are unlikely to expose any error, while increasing the number of "good" test cases that have a high probability of producing an erroneous output. The aim is to find minimal set of test cases that are likely to expose faults using mutated versions of the original program. In FAexGA approach, crossover probability varies according to the age intervals assigned during lifetime. Fuzzy logic controller (FLC) is used for determining probability of crossover. The FLC state variables include the age and lifetime of chromosomes (parents). The fuzzification interface of FLC includes variables that determine the age of an offspring. FLC assigns every parent values Young or Middle-age or Old. These values determine the membership for each rule in FLC rule base. The test cases relate to the inputs of tested software and are represented as a vector of binary or continuous values. The test cases are initialized randomly in the search space of possible input values. Genetic operators are applied and the test cases are evaluated based on the fault – exposing - capability using mutated versions of original program.

Chartchai Doungsa et al. [33] proposed GA-based test data generation technique to generate test data from UML state diagram, so that test data can be generated before coding. The test cases can be generated as per the specifications of the software. Specifications can be in the form of UML diagrams, formal language specifications or natural language description. Sequence of triggers for UML state diagram can be used as a chromosome. The sequence of triggers is an input for the state diagram which acts as test cases for a program to be tested. Test cases are selected based on their fitness function. Test case with best fitness value is selected as parents. Based on the fitness function the selection operator is used to apply crossover and mutation operator to the sequence of triggers. Crossover operator is then applied to the sequence of triggers. This operator then generates new states and transitions. After a new generation is created, UML state diagram is then executed again to check for new chromosomes. In this work, test cases are generated from UML state diagram so that test data can be generated before coding.

The effectiveness of test cases generated from the proposed fitness function is not evaluated with other test case generation techniques from the UML.

V. SOFT COMPUTING IN REGRESSION TESTING

Regression testing is type of testing carried out to ensure that changes made in the fixes or any enhancement changes are not impacting the previously working functionality. It is executed after enhancement or defect fixes in the software or its environment. It gives assurance that newly added features do not cause any problem or side effects in the functioning of the system. This is generally performed in maintenance phase of software development cycle.

Harsh Bhasin et. al [34] proposed Fuzzy Regression Expert System (FRES) which comprises of three components knowledge base, inference engine and user interface. Knowledge base contains all the rules. Inference engine takes the decision by checking which rules are satisfied by facts, prioritize the rules that are satisfied and execute the highest priority rules. The rules are to be prioritized based on premise discussed in the section. Inference engine processes the rules that are extracted and whose patterns are satisfied by facts in contention. The user interface presents the user available facts and other information as input.

VI. NEURAL NETWORK FOR SOFTWARE TESTING

Various blocks in a program are guarded by various branch predicates and the execution of the block depends on the predicate evaluations. Abhas Kumar [35] defined a function for each of the branch predicate. The program is instrumented so as to record the function evaluation for the corresponding inputs. A large record of such mappings from external inputs to the evaluation of branch functions can then be modeled using a neural network. Once the code is instrumented, data corresponding to the evaluation of branch predicates and the external inputs to the program is done is collected. He applies his approach to execute only a particular block of code, given all other blocks of the code have been executed considerable number of times. Within short number of such inputs, the predictor is able to guess a correct set of inputs which will execute the required statement. To speed up the process of test case generation inverse function was model by suing neural networks. He constructs such models for each of the given statements and found that the neural networks were able to learn fairly quickly and predict accurate values which in turn provide a good method to reduce the time required for test-data generation. His method demonstrated the use of neural networks in efficient test data generation.

VII. CONCLUSION

In this paper, applications of Soft Computing in different types of software testing are discussed. It is found that by using Soft Computing approaches for Software Testing, the results and the performance of testing can be improved.

REFERENCES

1. Dr. Velur Rajappa, Arun Biradar, Satanik Panda "Efficient software test case generation Using Genetic algorithm based Graph theory" International conference on emerging trends in Engineering and Technology, pp. 298-303, IEEE (2008).
2. Praveen Ranjan Srivastava and Tai-hoon Kim "Application of Genetic algorithm in software testing", International Journal of software Engineering and its Applications, vol.3, No.4, pp. 87-96 (2009).
3. André Baresel , Hartmut Pohlheim , Sadegh Sadeghipour, Structural and functional sequence test of dynamic and state-based software with evolutionary algorithms, Proceedings of the 2003 international conference on Genetic and evolutionary computation: PartII, July 12-16, 2003, Chicago, IL, USA
4. O. Buehler and J. Wegener. Evolutionary functional testing of an automated parking system. In International Conference on Computer, Communication and Control Technologies and The 9th International Conference on Information Systems Analysis and Synthesis, Orlando, Florida, USA, 2003.
5. H. P. Schwefel and R. Manner, editors, Parallel Problem Solving From Nature, pages 176–185. Springer-Verlag, October 1990
6. D E Goldberg, “Genetic Algorithms in Search, Optimization and Machine Learning”, Addison-Wesley, Reading, 1989.
7. J Holland, “Adaptation in Natural and Artificial Systems”, MIT Press, Cabmridge, MA, 1975.
8. D Koza, “Genetic Programming, On the Programming of Computers by Means of Natural Selection”, MIT Press, Cambridge, MA, 1992.
9. P von Laarhoven and E Aarts, “Simmulatd Annealing: Theory and Applications, Mathematics and its Applications” Kluwer, Dordrecht, 1987.
10. http://www.softcomputing.net.in/ (Access on 15th Feb, 2014)
11. Chattopadhyay S (2006), “Soft Computing Techniques in combating the complexity of the atmosphere-a review”, Arxiv preprint nlin/0608052.
12. David E. Goldberg (1989)” Genetic Algorithm in Search, Optimization and Machine Learning”, Pearson Education-India.
13. M. Melanie, “An Introduction to Genetic Algorithms, Massachusetts”, MIT Press, 1999.
14. K. F. Man , K. S. Tang and S. Kwong "Genetic algorithms: Concepts and applications", IEEE Trans. on Industrial Electronics, vol. 43, no. 5, pp.519 -534 1996
15. S. Sabharwal, R. Sibal and C. Sharma, "Applying genetic algorithm for prioritization of test case scenarios derived from UML diagrams", International Journal of Computer Science Issues (IJCSI), vol. 8, No. 2, pp. 433-444, May 2011.
16. Radim Belohlavek and George J. Klir. “Concepts and fuzzy logic”, Cambridge Mass. London: MIT Press, 2011
17. http://www.seattlerobotics.org/encoder/mar98/fuz/flindex.html, (Access on 18th Feb, 2014)
18. [18] Maureen Caudill, “Neural networks primer, part I”, AI Expert, v.2 n.12, p.46-52, Dec. 1987
19. Zupan, J. (1994) “Introduction to artificial neural network (ANN) methods: what they are and how to use them” Acta Chim. Slov. 41, 327–352.
20. Naresh Chauhan, “Software Testing: Principles and Practices”, Oxford University Press, 2010.
21. Paul C. Jogersen, “Software testing: A craftsman approach” 3rd edition, CRC presses, 2008.
22. http://khannur.com/stb6.2.htm (access date: 19th Feb, 2014)
23. Z. Bo and W. Chen, "Automatic generation of test data for path testing by adaptive genetic simulated annealing algorithm," in Computer Science and Automation Engineering (CSAE), 2011 IEEE International Conference on, 2011, pp. 38-42.
24. Praveen Ranjan Srivastava et. al., “Generation of test data using Meta heuristic approach” IEEE, 2008, pp.19 - 21.
25. Ribeiro, J. C. B., Zenha-Rela, M. A., and de Vega, F. F. (2008a), “A Strategy for Evaluating Feasible and Unfeasible Test Cases for the Evolutionary Testing of Object-Oriented Software” In Proceedings of the 3rd international workshop on Automation of Software Test (AST ’08) , pages 85–92, Leipzig, Germany. ACM.
26. Girgis, “Automatic test generation for data flow testing using a genetic algorithm”, Journal of computer science, 11 (6), 2005, pp. 898 – 915.
27. Yeresime Suresh et. al, “A Genetic Algorithm based Approach for Test Data Generation in Basis Path Testing” The International Journal of Soft Computing and Software Engineering, Vol. 3, No. 3, Special Issue [SCSE’13], March 2013
28. Premal B. Nirpal and Kale K.V.(2010), “Comparison of Software Test Data for Automatic Path Coverage Using Genetic Algorithm”, Internal Journal of Computer Science and Engineering Technology, Vol. 1, Issue 1.

29. Jasmine Minj Lekhraj Belchanden, "Path Oriented Test Case Generation for UML State Diagram using Genetic Algorithm" International Journal of Computer Applications (0975 – 8887) Volume 82 – No 7, November 2013
30. Eugenia Díaz , Javier Tuya , Raquel Blanco , José Javier Dolado, "A tabu search algorithm for structural software testing", Computers and Operations Research, v.35 n.10, p.3052-3072, October, 2008
31. Francisca Emanuelle et. al., "Using Genetic algorithms for test plans for functional testing", 44th ACM SE proceeding, 2006, pp. 140 - 145.
32. Mark Last et. al., "Effective black-box testing with genetic algorithms", Lecture notes in computer science, Springer, 2006, pp. 134 -148.
33. Chartchai Doungsaard, Keshav Dahal, Alamgir Hossain, and Taratip Suwannasart, 2007, "An Automatic Test Data Generation from UML State Diagram using Genetic Algorithm", The proceedings of the Second International Conference on Software Engineering Advances.
34. H. Bhasin, S. Gupta, M. Kathuria, "Regression testing using fuzzy logic", International Journal of Computer Science and Information Technology (IJCSIT), 4(2), pp. 378-380, 2013.
35. Abhas Kumar, "Dynamic Test Case Generation using Neural Networks", http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.97.3110&rep=rep1&type=pdf (Access on 20th Feb, 2014)

AUTHORS PROFILE

Bipin Pandey, is currently pursuing M. Tech. from Jodhpur National University, Jodhpur. He is currently working as an Assistant Professor at Vyas Institute of Engineering & Technology, Jodhpur. He is OCJP Certified. His main areas of research are the Genetic Algorithms, Data Mining and Software Engineering.

Rituraj Jain, is currently pursuing Ph.D. from Pacific University, Udaipur. He is currently working as an Associate Professor & Head at Vyas Institute of Engineering & Technology, Jodhpur. He is Life time member of ISTE Chapter and also Cloud U Certified. His main areas of research are the Genetic Algorithms, Data Mining and Software Engineering. He has presented numerous papers on these topics in International Journals and Conferences.