Case Study for Regression Testing using Genetic Algorithm

Birendra Kumar Saraswat¹, Binayak Parashar², Arun Kumar Takuli³, Samender Singh⁴

¹Assistant professor, GL Bajaj Institute of Technology and Management, Gr. Noida, UP, India. ²Ajay Kumar Garg Engineering College, Ghaziabad, UP, India. ³Hi-Tech institute of Engg. & Technology, Ghaziabad, UP, India. ⁴Mangalmay Institute of Engg. And Technology, Gr. Noida, UP, India.

e-mail: ¹birendrasaraswat@gmail.com

Abstract. For quality-assured information engineering software assessment has become critical. Software quality is defined as all the features and proprieties of a software system that can meet certain criteria to test a software product to know how good it is. Software testing is a necessary process for software development, and automating data production helps reduce time and expense. The design of software tests in today's industry is mostly focused on the experience of the experts, while the tools for test automation can be used only for pre-planned tests. The new investigation focuses on the safe, accurate and effective discriminatory regression testing. A selective algorithm for regression testing is considered secure and sound if the disclosing of test cases is not ignored. Precision that tests the sum of the algorithm moving through the test cases without alteration. The successful selection of these cases is one of the key issues in regression testing. Due to the utility of selective regression testing, an efficient algorithm is important due to the speed of test case selection. We use a formal language approach in this article, to explain the problems and to give tentative solutions. For the test case selection problem, we find the genetic algorithm. We demonstrate that this genetic algorithm that satisfies both secure and reliable standards take time t in general exponentially. In addition, a secure algorithm requires the most severe case of O (n²) time and O(n!) (location, in which n is as many statements as possible, and t is as many test cases as possible.

Keywords: Software testing, Genetic algorithm, Test case generation, Regression testing

1. INTRODUCTION

The cost of software errors is becoming more complex and increasingly incorporated in business and government processes. The most difficult problems, because many elements communicate unpredictable and unexpected. Complex machine failures are among the most difficult problems. Such highly connected systems may have intermediate or transient defects that are especially hard to diagnose. The present work is part of an automated test case generation process that concentrates on the use of a genetic algorithm (GA). The aim is to permit the generation of large volumes of test data focusing on any found mistakes. Such uncommon or underrepresented occurrences are the main focus of the genetic search process, though producing some test cases that explored the larger areas of good results. Only after extended execution, many errors emerge and the program emphasizes new and
repeated test cases. Our first work used genetic algorithms to construct test cases for the machine. But testing at device level tends to be the real strength of automated testing at large scales [1-3]. How to build large data sets to identify fault patterns on complex systems using genetic algorithms effectively? Testing will preferably take place in a complete fashion, leaving no question that the software created and tested is bug-free. Unfortunately, very few programs are so complex that rigorous testing is carried out. Because thorough research is not necessarily feasible, the development of test sets must tackle the problem of limiting the number of test cases. These methods are the most common methodological approaches used in generating non-exhaustive test sets (1) by analyzing the programmed internal structure (denoted structural case selection and white-box testing). It shows in Figure 1.

The test case can be seen by its input and expected output. However, to be more effective. If I d am a unique identifier used to trace the test case, the input will consist of a series of values that are passed as input parameters to cross the desired running path and/or check the desired features. Input is the same as input [4-5]. The performance is the predicted values and configuration specifies how the test condition is to be configured and how the test case can be exercised etc.

![Figure 1: Test Information Flow](image)

The continuation of the paper is structured as follows: Part 2 describes the related work on traceability and necessary tests; Part 3 presents a genetic algorithm model of test case selection. The findings of section 4 are eventually summarized in section 5.

2. RELATED WORKS

Five key problem areas were established in previous Job in the area of regression testing. There are trouble areas: (1) the question of review of the source code for software changes and their effects on software actions, (2) version management, (3) development of test cases, (4) collection of regression test cases and (5) nondeterministic run-time behavior. (The way in which test cases are preserved should also, however, be defined in the fields (2), (3) and (4) for the purposes of completion.) In (1), it is necessary to detect not only textual changes but also semantic modifications.

For example, by using a system representation graph, the identification of semantic changes, shown by Horwitz et al. [5] can be detected. We are aware that no research was done on the effect of changes on the temporal output, i.e., flows determined by system calls, task-interlacing events and task planning process, in relation to control-flow and data flow at system level. On the 2nd, Industry and academia have learnt to use software to handle multiple versions of source code. In the third segment, you must construct the initial collection of test cases. The test cases are initially chosen on the basis of the coverage criterion. In addition, in 4, In order to balance the capability to identify errors and the costs of choosing and applying the test cases, a regression test subset should be selected. In [6], it is impossible or difficult to decide if faults were corrected during test runs. The focus of multi-tasking software tests was on checking the order of organization procedures or of simultaneous measures in Java but in the knowledge of the author there were identical limited research findings on tests of the granularity of the interleaving programmed declaration [7-10].
3. TEST CASES SELECTION USING GENETIC ALGORITHM.

3.1 Unit Testing

We are focusing on a different form of software testing, called the testing unit, which separately measures each module or device. A unit may be a process, a class or a collection of closely related classes in object-oriented programs. Programmers normally undertake unit testing and are the basis for all other evaluations, for example integration testing and device testing. The testing consists generally of three elements: the design of test cases (or the collection of test data), the implementation of test cases and the comparison of performance data with the expected results. All three components must be standardized for full automation of testing.

Two popular test data collection strategies are the black-box approach and the white-box approach. In the black-box process, test data is only collected on the basis of the definitions of inputs and outputs, i.e. without understanding the internal workings of the system being evaluated. Basic knowledge of computer code in the White box approach is utilized to selection test data, for example control flow of the system or data flow property. The two methods are mutually compatible and are sufficient for full research. We use both methods in our work.

3.2 Regression Testing

During the software development process, regression testing is done at various stages. Testing can expose problems during the whole development process and at all stages. These bugs are most often fixed through modifications to the source code, which lead to a need to check the program again. Therefore, there has arisen a need for regression testing. "Selective restart of the system or component to check that changes have not led to unintended results and the system or component still meets its defined specifications. Software glossary describes regression testing as” [11]. The regression test can be divided into two groups on the basis of the form of modification [12] Modifications to the source code cause corrective regression testing, although gradual regression testing is initiated by modifications to specifications. Regression testing for single-tasking applications (i.e., very structured applications) has been deliberately suppressed. The focus of research in this field was on the issue of regression testing. This is because a careful selection of tests will minimize research efforts dramatically. The basic concept is to pick test cases to re-test only modified components (as non-modified components remain valid). This problem is not straightforward since the question of how changes spread and affect unchanged sections of the code is difficult to decide [13-17].

3.3 Genetic Algorithms

In order to produce object-orientate test data automatically we use an evolutionary approach called genetic algorithms. Genetic algorithms are rooted in development mechanisms and natural genetics and exploit a possible optimization or search problem population [8-9]. Genetic Algorithms A series of possible solutions, consisting of a sequence of genes, is defined as chromosomes which are similar to the genetic material in nature. Each trial data can be seen as a chromosome [18-21]. In comparison with other solutions in the population, each solution is related to the fitness value that shows how good it is. Check coverage for fitness can be used in training. Fitness, as in nature, is the driving force behind better survival strategies. This ensures that new potential solutions with characteristics that are heisted from each parent solution are accomplished with selective breeding. Generations of successive organisms, called genetic operations, including crossover and mutation, are created. The goal is to make it possible for people to agree on a global solution. The fitness value of each solution is determined randomly generated solutions from an initial population. The members of a community are chosen to become parents based on their fitness values. In conjunction with the crossover operator, the parents form the offspring that concentrates the genes in two chromosomes. The crossover objective is producing offspring with both parents' best traits, resulting in better offspring. Many of the descendants are then transformed into demographic diversity. Then, from the young people and the old population, we chose a new generation. When terminations are achieved, the generational cycle must be repeated. While genetic algorithms are relatively new for research, the approach has proved successful. Genetic algorithms are also used to solve the problems of population growth in the context of a collection of restrictions in the search for a viable result [22-30]. The genetic algorithm is used in
this current application however; the principle of goodness relies on the outcomes of previous test
cycles preserved in a fossil record, to produce good test cases. In other words, an efficient correlation
is used instead of absolute fitness [31].

**Genetic Algorithm:**

1. **Step-1** Begin
2. **Step-2** $t \leftarrow 0$, Initialize $P(t)$
3. **Step-3** Evaluate $P(t)$
   while (not termination condition) do
   
   Begin $t \leftarrow t+1$
   
   **Step-4** select $P(t)$ from $P(t-1)$ according to evaluation
   
   crossover $P(t)$ according to crossover rate
   
   mutate $P(t)$ according to mutation rate evaluate $P(t)$ end

   **Step-5** End

**Genetic Algorithm Structure**

4. **EXPERIMENTAL RESULTS**

The following section describes the empirical family of studies, including the architecture, topics,
measurements and findings. The question of priority test cases and empirically analyzed various
priority techniques. These strategies were focused on the inspection of either software statements or
branches and defects. The relation is made to the other Greedy algorithm and the genetic algorithm to
obtain better results for the test cases [32-35]. For the test cases range.

It has been applied to research programs in the hope of demonstrating the applicability of the
proposed genetic algorithm-based test case generation techniques [36].

4.1 Case Study-1: Temporal Database for Data Base Management System

The key principle of this project is the calling and implementation of a temporal natural language
interface (TNLP) as a user for communicating with the temporal database system and simplifying the
query processing in a temporal database system. The framework is built with java, which can be run in
any operating system supporting java virtual machine and tested with banking domain data [37-39]. A
time-based Natural Language querying framework has been developed for users who are not using
SQL, and for intermediate users wishing to retrieve data from temporal databases without having to
learn query mechanisms such as SQL.

We have numerous modules like In this project
- Design Wizard
- Natural Language
- Queries
- Create Table
- View Table
- Update Table
- Delete Table
- Insert Table

Though the project has many modules and screens here we are presenting only one screen as
Sample Output (Design Form)

This is design wizard form. In which user has to give query by select, it as a natural language
processing form, in which we can give user query in a normal English language not all like a queries
in a SQL.

4.2 Performance Analysis of Genetic Algorithm with Case Studies

Genetic algorithms have higher efficiency than any other algorithm. The Proposed Genetic
algorithm is of finding the suitable test cases focused on application of regression. Each and every test
case contains some statements. These statements have some unidentified errors. The error mentioned
which are based on the test cases. If particular test cases contains the error is higher than other tests
cases then select the test cases. The selected test case is given the first priority. If test cases contain
same number of errors then select the first test case. In this way, finding the error is simple one and consume less time, Our Genetic algorithm overcomes all other algorithm.

| S. No. | Test Case Name       | No. of Statements | No. of Errors |
|--------|----------------------|-------------------|---------------|
| 1      | Query Generate Option| 29                | 20            |
| 2      | Query Result Option  | 27                | 12            |

5. CONCLUSIONS

Throughout this study we have highlighted that multiple task real-time regression testing contains two major problem areas: the problem of test selection and the problem of reproducibility. In response to selecting the right combination of inputs and outputs to evaluate updated programs, the test selection problems often include the selection of the required sequence. As far as reproductively is concerned, the issues addressed include practical difficulties in the re-testing of test cases. This involves the recurrence of tasks between sequences, the device control and data flow rates as well as of tasks. While academic results improve productivity and time-consuming regression testing methods and techniques, there are other benefits. Automation of the test activities could be more appropriate for the industry, in order to identify problems earlier in software development, since they could be conducted during working hours.

REFERENCES

[1] G. Antoniol, L. C. Briand, M. D. Penta, and Y. Labiche. A case study using the round-trip strategy for state-based class testing. In ISSRE ’02. IEEE Computer Society, 2002.
[2] Baresel, D. Binkley, M. Harman, and B. Korel. Evolutionary testing in the presence of loop-assigned flags: a testability transformation approach. In ISSTA, pages 108–118, 2004.
[3] Goyal, M. and B. Gupta. Study of shape, size and temperature-dependent elastic properties of nanomaterials. Modern Physics Letters B, 2019. 33(26): p. 1950310.
[4] Singh, P., Sharma, K., Kumar, A., & Shukla, M. (2016). Effects of functionalization on the mechanical properties of multiwalled carbon nanotubes: A molecular dynamics approach. Journal Of Composite Materials, 51(5), 671-680
[5] R. V. Binder. Testing Object-Oriented Systems: Models, Patterns, and Tools, Object Technology. Addison-Wesley, 1999.
[6] Goyal, M. and M. Singh, Size and shape dependence of optical properties of nanostructures. Applied Physics A, 2020. 126(3): p. 1-8.
[7] Singh, P., & Sharma, K. (2018) Mechanical and Viscoelastic Properties of In-situ Amine Functionalized Multiple Layer Graphene /epoxy Nanocomposites. Current Nanoscience, 14(3), 252-262.
[8] G. Booch, J. Rumbaugh, and I. Jacobson. The Unified Modeling Language User Guide. Addison-WesleyLongman, 1999.
[9] Sharma, K., & Shukla, M. (2014). Three-Phase Carbon Fiber Amine Functionalized Carbon Nanotubes Epoxy Composite: Processing, Characterisation, and Multiscale Modeling. Journal Of Nanomaterials, 2014, 1-10.
[10] L. C. Briand, M. D. Penta, and Y. Labiche. Assessing and improving state-based class testing: A series of experiments. IEEE Trans. Softw. Eng.,30(11):770–793, 2004.
[11] Kumar, A., Sharma, K., & Dixit, A. (2018), A review of the mechanical and thermal properties of graphene and its hybrid polymer nanocomposites for structural applications. Journal of Materials Science, 54(8), 5992-026.
[12] P. Chevalley and P. Th’evenod-Fosse. Automated generation of statistical test cases from UML state diagrams. In COMPSAC ’01, pages 205–214. IEEE Computer Society, 2001.
[13] Singh PK, & Sharma K, Molecular Dynamics Simulation ofGlass Transition Behaviour ofPolymer based Nanocomposites, Journal of Scientific & Industrial Research, 77 (10) (2018) 592-595.
[14] T. S. Chow. Testing software design modeled by finite-state machines. IEEE Trans. Softw. Eng., 4(3): 178–187, 1978.
[15] Kumar, A., Sharma, K., & Dixit, A. (2019). Carbon nanotube- and graphene-reinforced
multiphase polymeric composites: review on their properties and applications. Journal Of Materials Science, 55(7), 2682-2724.

[16] P. Fröohlich and J. Link. Automated testcase generation from dynamic models. In ECOOP ’00, pages 472–492. Springer-Verlag, 2000.

[17] Manoj Kumar Shukla, & Kamal Sharma. (2019). Effect of Carbon Nanofillers on the Mechanical and Interfacial Properties of Epoxy Based Nanocomposites: A Review. Polymer Science, Series A, 61(4), 439-460.

[18] D. E. Goldberg and K. Deb. A comparative analysis of selection schemes used in evolutionary algorithms. In FOGA, pages 69–93, 1990.

[19] Kumar, A., Sharma, K., & Dixit, A. (2019). A review on the mechanical and thermal properties of graphene and graphene-based polymer nanocomposites: understanding of modeling and MD simulation. Molecular Simulation, 46(2), 136-154.

[20] F. Ipate. Testing against a non controllable stream Xmachine using state counting. Theoretical Comput. Sci., 353(1-3):291-316, 2006.

[21] Mausam, K., Sharma, K., Bharadwaj, G., & Singh, R. (2019) Multi-objective optimization design of die-sinking electric discharge machine (EDM) machining parameter for CNT-reinforced carbon fibre nanocomposite using grey relational analysis. Journal Of The Brazilian Society Of Mechanical Sciences And Engineering, 41(8).

[22] B. Jones, H. Sthamer, and D. Eyres. The automatic generation of software test data sets using adaptive search techniques. In Proceedings of 3rd International Conference on Software Quality Management, pages 435-444, 1995.

[23] Shukla, M., & Sharma, K. (2019) Improvement in mechanical and thermal properties of epoxy hybrid composites by functionalized graphene and carbon nanotubes. Materials Research Express, 6(12), 125323.

[24] D. Lee and M. Yannakakis. Principles and methods of testing finite state machines - A survey. In Proceedings of the IEEE, volume 84, pages 1090-1126, 1996.

[25] Y.-S. Ma, J. Offutt, and Y. R. Kwon. Mujava: an automated class mutation system. Softw. Test. Verif. Reliab., 15(2):97–133, 2005.

[26] Kumar, K., Sharma, K., Verma, S., & Upadhyay, N. (2019) Experimental Investigation of Graphene-Paraffin Wax Nanocomposites for Thermal Energy Storage. Materials Today: Proceedings, 18, 5158-5163.

[27] P. McMinn. Search-based software test data generation: a survey. Softw. Test., Verif. Reliab., 14(2): 105–156, 2004.

[28] Goyal, M. and B. Gupta, Analysis of shape, size and structure dependent thermodynamic properties of nanowires. High Temperatures--High Pressures, 2019. 48.

[29] P. McMinn and M. Holcombe. The state problem for evolutionary testing. In GECCO, pages 2488-2498, 2003.

[30] P. McMinn and M. Holcombe. Evolutionary testing of state-based programs. In GECCO, pages 1013-1020, 2005.

[31] B. Meyer. Design by contract. IEEE Computer, 25(10):40-51, 1992.

[32] Z. Michalewicz. Genetic algorithms + data structures = evolution programs (3rd ed.). Springer-Verlag, London, UK, 1996.

[33] M. Mitchell. An Introduction to Genetic Algorithms. MIT Press, Cambridge, MA, USA, 1998.

[34] J. Offutt and A. Abdurazik. Generating tests from UML specifications. In UML, pages 416-429, 1999.

[35] J. Offutt, S. Liu, A. Abdurazik, and P. Ammann. Generating test data from state-based specifications. Softw.Test., Verif. Reliab., 13(1):25-53, 2003.

[36] R. P. Pargas, M. J. Harrold, and R. Peck. Test-data generation using genetic algorithms. Softw. Test., Verif. Reliab., 9(4):263–282, 1999.

[37] P. Tonella. Evolutionary testing of classes. In ISSTA, pages 119–128, 2004.

[38] N. Tracey, J. Clark, and K. Mander. Automated program flaw finding using simulated annealing. In ISSTA ‘98, pages 73–81. ACM Press, 1998.

[39] N. Tracey, J. A. Clark, K. Mander, and J. A. McDermid. An automated framework for structural test-data generation. In ASE, pages 285–288, 1998.