

Genetic Algorithm Approach to Optimize Test Cases

Baswaraju Swathi^{#1}, Dr.Harshvardhan Tiwari^{*2}

[#] Research Scholar, Department of CSE, Jain University, Bengaluru, Karnataka, India.
² Associate Professor, CIIRC, Jyothy Institute of Technology, Bengaluru, Karnataka, India

¹baswarajuswathi@gmail.com
²tiwari.harshvardhan@gmail.com

Abstract - Test case generation is considered a significant and complex aspect in software testing, ensuring the quality of a software product. In a highly competitive environment, web applications have become crucial to most enterprises, demanding the application's quality. The characteristics of such applications include client-server, distributed, and dynamic. Hence meticulous testing of a web-based application becomes necessary. Many strategies have been proposed to address the issues w.r.t test case generation for web applications. One such strategy is a Genetic Algorithm (GA), which is an evolutionary technique. In this paper, we analyze the test suit generation as a complex problem and derive the test cases with traditional test case generation approaches where the common generation problems are addressed with a case study.

Further Genetic Algorithm approach, the parameters which can enhance the test case generation is proposed. The various encoding and selection techniques of GA are considered. The parameter of fitness function which determines the success of GA is analyzed. Finally the evaluation criteria code coverage is examined to assess the test effectiveness.

Keywords — code coverage, evolutionary technique, fitness function, Genetic Algorithm, test case generation

I. INTRODUCTION

Test case/suite generation is the most central and essential practice of software testing. There are numerous methods accessible for creating test cases. The goal-oriented test case creation approach is to wrap a specific segment, statement, or procedure. The execution path is not relevant here, but the primary objective is to test the target. The random approach builds test cases assuming errors and system faults. The specification model builds test cases based on the formal requirement specifications [1]. The code-dependent case creation method generates the source code's control flow path, derives the tests accordingly, and tests the executed flow. The other type of test case generation method uses the UML diagram to build the test cases. Many other test case generation methods are available in the testing. Irrespective of the approach, an appropriate test case generation process is one of the most decisive factors for implementing a successful project.

Testing of web applications [2] remains a challenging task for the testers. The type of browsers, interfaces, security coercion, and overall application integration are among the issues. As testing is a critical phase in the software development process, the web application developer expects unforeseen issues connected to the web application and the testing progression [3]. Every key problem, coupled with testing, can run into infinite correlated issues, which can usually be solved if appropriately acknowledged. Web application testing challenges include: Integration testing presents problems with implementations between distinct system components before deployment. Besides, integration testing can show the various issues that the program might have while communicating with the other applications, enabling the programmer to modify things. System and network inconsistency, specific interaction models, and overall performance are just a few of the problems related to deployment testing. Interoperability is asserting end-to-end compatibility among communications networks is often a demanding challenge. Applications use different browsers and operating systems. To collect data, it is very important to test each one to validate a clear information path. Though the browsers are identical, the web application may use different screen resolutions and different hardware and software configurations, thereby creating severe applications. Security is an important parameter. Tests w.r.t cyber threats should be countered and neutralized. Usually, such types of threats lead to data loss or data manipulation. The main challenge in security in web applications deals with unsecured communication, malicious file integration, unauthorized and unauthenticated procedures on the client, and server-side. Performance is all about timing. The web applications that take much time to load/execute are usually not recommendable. Integrating and interoperability issues, the hardware support, and extending application features are the main issues w.r.to the mentioned testing. Usability-Consistency is considered crucial as the variation in browsers, scalability, and interactivity issues need to be considered. The web app needs to be used effectively.



II. TEST CASE GENERATION

The test derivation can be considered as a single objective where the priority can be given to coverage, which includes structural coverage [4], all definitions coverage, all path coverage [5], and code coverage [6,7] is an important metric to evaluate test effectiveness. The other factors can be Requirement change volatility, which may lead to an increase in the testing effort, and the rate of change may lead to cost and time effort. Test case optimization also has ranked the test cases, parallel execution, independent and dependent on other test cases may be in a single module or different modules, thereby defining multiple objectives [8]. The various methods like weighted sum $W(X)=\sum W_i X_i$ and Epsilon constraint method $f_m(X) \leq \epsilon$ is considered to derive the functionality. A test suite is a complete suite of test cases usually represented as:

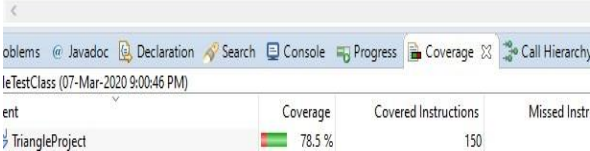
$T_s = \{ \langle t_1, tp/f_1 \rangle, \langle t_2, tp/f_2 \rangle, \langle t_3, tp/f_3 \rangle, \dots, \langle t_n, tp/f_n \rangle \}$, T_s –test suite, t_d -test data and tp/f_1 result of the test case passed/failed. Considering the most popular triangle problem, the figures depict the importance of code coverage as an evaluation parameter.

Sample of unit test cases:

```
@Test
public void test1()
{
    Triangleclass1 c=new Triangleclass1();
    String expectedval= "Scalene";
    String actualval= c.ftr(2,3,4);
    assertEquals(expectedval,acualval);
}

@Test
public void test2()
{
    TriangleClass c=new TriangleClass();
    String expectedvalue="equilateral";
    String actualvalue=c.ftringle(1,1,1);
    assertEquals(expectedvalue,actualvalue);
}

@Test
public void test3()
{
    TriangleClass c=new TriangleClass();
    String expectedvalue="isosceles";
    String actualvalue=c.ftringle(5,5,1);
    assertEquals(expectedvalue,actualvalue);
}
}
```



ent	Coverage	Covered Instructions	Missed Instr
TriangleProject	78.5%	150	

Fig 1: Code coverage

The above figure shows that for the test cases executed for the triangle problem, the code coverage is 78.5%. To improve the code coverage, we have to run the other test cases as well. Hence code coverage can be considered. The

below section deals with the test case generation usually applicable for a web application. Considering the following application, a shopping cart app, considering the app's various scenarios, the test cases can be categorized as mentioned below as per random testing alone. Exhaustive testing is usually not preferred due to the budget and time constraints and several other factors. If specifically, the test cases should deal with black box and white box testing scenarios. For example, under black box testing equivalence testing, the test classes are divided into classes considering both positive and negative cases must be specified. Decision table testing specifies the scenarios with input combinations, and behavior is captured. T-way testing [9] is considered a superior testing technique as it drastically reduces the test case combination. As the paper discusses the web applications, each page in web application consists of elements; therefore, the combination of the input values itself generates a huge amount of test cases, considering the functionality of every element will further increase the count of test cases whereby there is a need for test optimization. The following section describes the unit test cases written in selenium web driver, on a web application "newtours.demoaut.com," with test data as "<autotest>" "<autotest>" ">" testing the login functionality of the website.

Fig 2: Selenium unit test cases

```
public void LoginStudent() {
    WebDriver driver = IDriver.Launch("http://newtours.demoaut.com/");

    WebElement objUserLoginNameTextBox = driver.findElement(By.name("userName"));
    objUserLoginNameTextBox.sendKeys("autotest");

    WebElement objUserPasswordTextBox = driver.findElement(By.name("password"));
    objUserPasswordTextBox.sendKeys("autotest");

    WebElement objSubmitButton = driver.findElement(By.name("login"));
    objSubmitButton.click();

    driver.manage().timeouts().implicitlyWait(5, TimeUnit.SECONDS);

    WebDriverWait driverWait = new WebDriverWait(driver, 10);

    driverWait.until(ExpectedConditions.visibilityOf(driver.findElement(By.name("findFlights"))));

    assertEquals("Find a Flight: Mercury Tours:", driver.getTitle());

    IDriver.Close();
}
```

III. METHODOLOGY

Genetic Algorithm is an optimization algorithm [10,11] based on natural evolution. GA deals with the natural way of selecting the best population, usually represented with the fitness function. Every generation is expected to produce the best fit of individuals as parent chromosomes evolve to be best; offspring also evolve. Fitness function is the critical evaluation factor of GA as it determines the best fit of individuals. The iteration can be stopped based on the constraints of the optimization problem. Crossover

operation leads to generating a new population, and mutation operation usually balances the diversity and convergence[12].

The selection stage [13] regulates the individuals chosen for merging and the number of offsprings the selected chromosome generates. The foremost standard of selection approach is the better individual is selected as a parent, determining which chromosomes need to be considered for the next iterations and stopping, hence ensuring the best solutions. Roulette wheel selection attaches a probability to an individual such that the probability of individually selected is $P_{ti} = \text{fitness } i / \sum i(\text{fitness } k)$ for $k = 1$ to the number of individuals considered as a whole which can be implemented using Naive Algorithm, which calculates the total of all fitness, generate a number which is random $\text{sum} > r$ the iteration stops. Rank Selection considers relative fitness, unlike Roulette wheel, which considers absolute, i.e., Population of N solutions rank $N, N-1, N-2,$ and so on till 1 is considered. Tournament Selection selects individuals at random the best fitness, usually called as the winner is chosen for crossover. Crossover operations promote diversity. The finding of global solutions Multi-Point Crossover simplifies the one-point crossover wherein discontinuous segments are swapped to get a new population. Uniform crossover each gene is considered separately. The mutation is the part of the GA that is associated with searching the search space, which is necessary to the convergence of the GA. Bit flip identifies arbitrary bits and flips the bits, used commonly in binary encoded GA. Random Resetting is an addition of the bit flip for the numeral depiction. A random value from the set of acceptable values is initialized to an arbitrarily selected gene. Swap Mutation selects two points on the chromosome at arbitrary, substitute the values. They are used in permutation-based encodings. Fitness functions [14] depend on the goal, priorities, and execution specifications. Usually, a weighted sum is used, as mentioned in previous sections.

A. GA APPROACH

As per the above mentioned there are several characteristics of web applications which makes the test case generation a complex process. GA approach here considers the initial population form the derived testcases, $T_s = \{ \langle td1, tp/f1 \rangle, \langle td2, tp/f2 \rangle, \langle td3, tp/f3 \rangle, \dots \langle tn1, tp/fn \rangle \}$.

The idea is to optimize the above test cases with a GA algorithm run on the mentioned. White box testing the metrics considered will be structural coverage criteria and data flow coverage criteria[15].

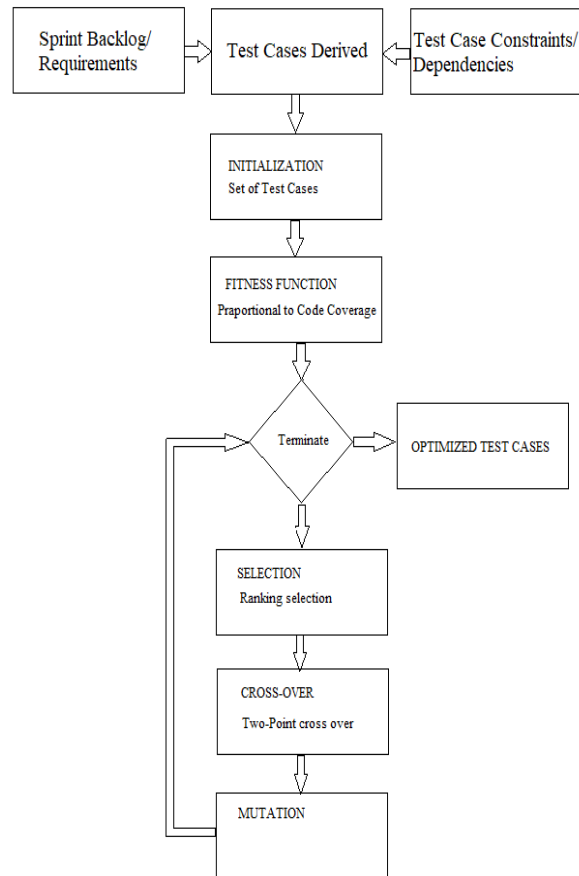


Fig 3: GA Approach

The problem of test case generation is given by $T_1, T_2, T_3 \dots T_N$ -TEST SUIT,

$F(T) = \{T_1, T_2, T_3, \dots T_N\}$, find T such that $T = \{t_1, t_2, t_3 \dots t_n\}$ set of test cases in $\{T_1, T_2, T_3, \dots T_N\}$ is the best solution which gives maximum coverage.

The encoding scheme considers representing the initial population as a node. A page can be specified with URL and a pointer were pointing to the next page. Request can be specified with the filed name and value of the field it carries as the page can link to the other page bypassing some information; Ranking based selection is used as described in the above section.

Consider the following use case.

```

<useCase id="Shopping Cart ">
  <description_usecase>Login.jsp page redirects to Shopping cart page for a user. </description_usecase>
  <Sequence>
    <step id="No1"> User selects the books based on the description fields available</step>
    <step id="No2"> Adds them to the cart </step>
    <step id="No3">the system adds the list of the items to the cart where user can edit further </step>
    <step id="No4">System displays the summary </step>
  </Sequence>
</info>
  
```

```

<Steps>
  <step id="1.1"> If there are no books as per the
  description is empty system displays a message
  and ends use case. </step>
</Steps>
</info>
</useCase>

```

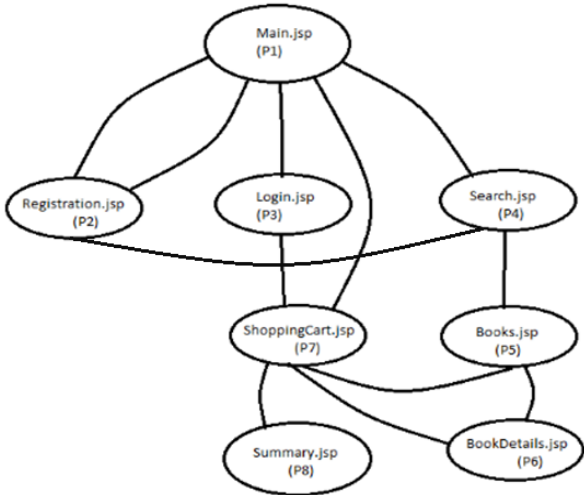


Fig 4: A Sample Application

Test Scenarios considered for the above mentioned infig.6. ofweb application page,
 C1:P1->P4->P5->P7,
 C2:P2->P1->P3->P7->P8
 C3: P2->P1->P3->P4->P5->P6,
 C4: P1->P2->P1->P7->P8,
 the same can be considered even for a Triangle problem. (S1->S2->S3->S4), where S is the statements which further include statement coverage and branch coverage.

The fitness function derived here is the value proportional to code coverage. Fitness function is considered to be a weighted sum of the objective functions of data dependencies and link dependencies and the constraints on a web application which are usually applicable,

$$F_i = \sum 1/C_i(f_i) * d_i, i\{t_1, t_2, t_3..t_n\}$$
 where f_i is the fitness of individual chromosome proportional to code coverage of the selected chromosome and d_i is dependency factors for both unit testing and black-box testing. Dependency factors for the Triangle problem considered here, as stated above.

Cross over: TwoPoint Cross over to get the new population used to set the maximum combinations from the selected chromosomes as it increases diversity as described in fig.3.

Mutation: Mutation will transform the chromosomes, thereby removing the duplicate test cases and creating variant test cases.

Acceptance and stop criteria usually depend on the application-specific and many other local constraints. The

current work stops the number of iterations after achieving a specific code coverage percentage.

Step-1:Initialization:C1,C2,C3,C4

STEP-2:FitnessAssignment:directly proportional to code coverage

Step-3:Selection: based on the Fitness value

Step-4:Crossover:

C1:P1->P4->P5->P7

C2:P2->P1->P3->P7->P8

C11:P1->P3->P7->P8

C21:P2->P4->P5->P7

STEP-5:Mutation:

C1:P1->P4->P5->P7

C3:P1->P2->P1->P7->P8

C13:P1->P2->P1->P4->P5->P7->P8

IV. EVALUATION OF THE GA

The system under test from the open-source was considered, and our GA an optimized algorithm was applied.GA executed has resulted in a Pareto set intended to maximize the objectives[17]. The change in the average objective results was recorded. Sampling the approximate values in the test suite achieved considerable fitness and code coverage values.

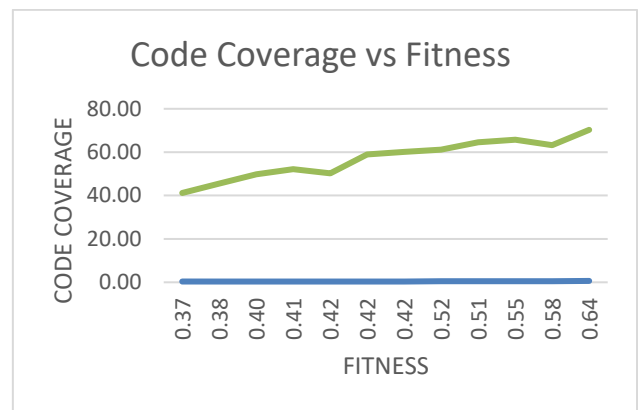


Fig 5: Code Coverage vs. Fitness

V. CONCLUSION

This paper gives an insight into the test case generation process-specific of web applications testing and White Box testing specific to the Triangle problem. The properties of web application which play a significant role in test case generation were specified. Generation of Test cases as a multipurpose optimization is addressed; a Genetic Algorithm, an evolutionary technique that can be applied to test case generation, was analyzed, and the steps in the process. Code coverage, an important metric in deciding the test effectiveness, was focused and measured with an example module. GA Methodology is proposed to be the major soft computing technique to apply in the implementation of test design development across the software test projects.

REFERENCES

- [1] NichaKosindrdech and JirapunDaengdej, 2010, "A Test Case Generation Process and Technique," Journal of Software Engineering,4:265-287, DOI: 10.3923/jse.2010.265.287.
- [2] Nabuco M., Paiva A.C.R. (2014), "Model-Based Test Case Generation for Web Applications," In Murgante B. et al. (eds) Computational Science and Its Applications – ICCSA 2014. ICCSA 2014. Lecture Notes in Computer Science, vol 8584. Springer, Cham. https://doi.org/10.1007/978-3-319-09153-2_19
- [3] Arora A., Sinha M, "Web Application Testing: A Review on Techniques, Tools, and the State of Art," International Journal of Scientific & Engineering Research, Volume 3, Issue 2, February-2012 I-ISSN 2229-5518.
- [4] Arun Sharma, Rajesh Kumar, "Towards Multi-Faceted Test Cases Optimization," Journal of Software Engineering and Applications 4:550-557 · January 2011.
- [5] M. Farina, K. Deb, and P. Amato, "Dynamic multiobjective optimization problems: test cases, approximations, and applications," IEEE Transactions on Evolutionary Computation, vol. 8, no. 5, pp. 425-442, Oct. 2004, doi: 10.1109/TEVC.2004.831456.
- [6] V. Antinyan, J. Derehag, A. Sandberg and M. Staron, "Mythical Unit Test Coverage, in *IEEE Software*," vol. 35, no. 3, pp. 73-79, May/June 2018, doi: 10.1109/MS.2017.3281318.
- [7] BaswarajuSwathi, Harshvardhan Tiwari, "Test Case Generation Process using Soft Computing Techniques," International Journal of Innovative Technology and Exploring Engineering (IJITEE), ISSN: 2278- 3075, Volume-9 Issue-1, November 2019.
- [8] A. Shahbazi and J. Miller, "Black-Box String Test Case Generation through a Multiobjective Optimization" in IEEE Transactions on Software Engineering, vol. 42, no. 04, pp. 361-378, 2016.doi: 10.1109/TSE.2015.2487958
- [9] Rozmie R. Othman, Kamal Z Zamli, "T-Way Strategies and Its Applications for Combinatorial Testing," International Journal on New Computer Architectures and Their Applications(IJNCAA).
- [10] HirohideHaga, Akihisa Suehiro, "Automatic test case generation based on genetic algorithm and mutation analysis," 2012 IEEE International Conference on Control System, Computing and Engineering.
- [11] Shirole M., Kumar R, "A Hybrid Genetic Algorithm Based Test Case Generation Using Sequence Diagrams," In Ranka S. et al. (eds) Contemporary Computing. IC3 2010. Communications in Computer and Information Science, vol 94. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-14834-7_6
- [12] Deepak Kumar, ManuPhogat, "Genetic Algorithm Approach For Test Case Generation Randomly: A Review," International Journal of Computer Trends and Technology (IJCTT) – Volume 49 Number 4 July 2017.
- [13] A. Shukla, H. M. Pandey, and D. Mehrotra, "Comparative review of selection techniques in genetic algorithm," 2015 International Conference on Futuristic Trends on Computational Analysis and Knowledge Management (ABLAZE), Noida, 2015, pp. 515-519, doi: 10.1109/ABLAZE.2015.7154916.
- [14] ArinaAfanasyeva, Maxim Buzdalov, "Choosing Best Fitness Function with Reinforcement Learning," 2011 10th International Conference on Machine Learning and Applications and Workshops.
- [15] ShvetaParnami, Krishna Swaroop Sharma, "Empirical Validation of Test Case Generation Based on All-Edge Coverage Criteria," International Journal of Computer Applications, September 2015.
- [16] Panichella, A., Kifetew, F. M., &Tonella, P. (2018). "Automated Test Case Generation as a Many- Objective Optimisation Problem with Dynamic Selection of the Targets," IEEE Transactions on Software Engineering, 44(2),122–158.