International Journal of Computer Applications (0975 – 8887)
Volume 136 – No.13, February 2016

Test Cases Generation on Robotics for basis Path Testing using Genetic Algorithm

Anju Bala
Research Scholar
Maharshi Dayanand University, Rohtak

Rajender Singh Chhillar, PhD
Professor in Department Of Computer Science & Applications
Maharshi Dayanand University, Rohtak

ABSTRACT
The paper explores the Genetic Algorithm approach to generate adequate and accurate test data for a specific target path. Software plays an important role in many of the systems, where the usage of software for a variety of purposes in different domains of modern life is rapidly increasing. With advancements in technology, it becomes quite complex whereas, software often contains errors. So testing consumes more money and time, which leads to automation that reduces human effort in finding bugs and errors. Actually, Automation in the last phase of system development is similar to manual testing. In both cases, bugs are detected only after code has been complete. Software testing is the most important component of software development process. Path testing is a popular structural testing method which uses the source code of a program to find every possible executable path. Test data generation is a key problem in software testing and its automation will improve the efficiency and effectiveness of software testing. Genetic Algorithm is an adaptive heuristic search algorithm that premise on evolutionary ideas of natural selection and genetic. In this paper, Genetic Algorithm is used to generate path by converting the program into its corresponding Control Flow Graph and then automatically generates the test data for the target path using different sets of GA operators.

Keywords
Testing Techniques, Genetic Algorithm, Case Study and Path Testing, Conclusion

1. INTRODUCTION
Software testing is a process to analyze the software to detect the bugs, fault and failures; it is used to validate the features of the software, and is used to ensure that it satisfies the customers and developers requirements. Mainly there are three approaches in software testing namely, black-box testing, white-box testing and grey box testing. Black-box testing is testing the functionality against software specifications. Testers determine what input should be given, what output should be generated, and testers analyze the external behaviour of the software. On the other hand, white-box testing is an examination of the logic, and the procedure used in the software. It focuses on testing the data structures, branches, loops, conditions, objects, messages, critical paths generated. Grey-box testing is a combination of both white and black box testing. In grey box testing tested doesn't have complete information of the system but has some idea about the system. So grey box testing is not completely white box or black box testing Software testing is a verification process which promises the clients expectations and requirements about the software. Verification is done to guarantee that developed software is meets clients requirements.

1.1 Testing Techniques
1.1.1 Black Box Testing
Black box testing is mainly used to execute the requirements of the system. Black-Box testing examines functionality without knowledge of internal implementation. It mainly concentrates on whether the input is accepted and output is generated or not. It concentrates on functional requirements so it is also called as functional testing. Functional testing evaluates the correctness of the program without any knowledge of how the software is implemented. In black box testing, testers test software through user interfaces, data structures, data base, or the application programming interfaces at the later stages of software development.

1.1.2 White Box Testing
White box testing known as clear box testing, glass box testing, transparent box testing and structural testing. In White box testing an internal perspective of the system, as well as programming skills are used to design test cases. White box testing is detailed examination of the code. Code coverage criteria is defined using segment coverage, branch coverage, node testing, statement coverage, condition coverage, basis path testing, data flow testing, path testing and loop testing. White box testing can be applied at the unit integration and system levels of software testing process. The test procedure attempts to execute every part of the source code using the test data.

1.1.3 Grey Box Testing
Grey-box testing is a combination of benefits of both black box-testing and white box-testing. Grey box testing involves having knowledge of internal data structures and algorithms for designing tests, while executing those tests at the user or black-box level. As with black-box testing, grey-box testing uses a specification for creating test cases. In grey box testing the tester has limited knowledge of the system. Grey-box testing implements intelligent test scenarios, based on limited information. The specification used in grey-box testing does not specify only the requirements of a system, but it also describes the behavior of the system.

2. DATA COLLECT FROM INDUSTRIES (WITHOUT GENETIC ALGORITHM)
The Robotics & Mechatronics division covers very large areas. It is difficult to point out the key parameters for looking over and predicting the comprehensive technical development and discuss the future of these technologies. Thus, in this roadmap, we decided to focus on the industrial robots that have been increasing the social and technical importance. They are automatically controlled, reprogrammable multipurpose manipulator, programmable in three or more

axes, which may be either fixed in place or mobile for use in industrial applications. In this research paper we discussed that Metal detector Robot; we will find finest result with the help of Genetic algorithm as compare to Random selection of best cases.

i. Mean power rate density

ii. Accuracy

iii. Intelligence level

3. GENETIC ALGORITHM IN ROBOTICS

First of all Evolutionary Robotics which used the concept of Genetic Algorithm. It is the Methodology that uses Evolutionary Computation to develop controllers for autonomous robots. Algorithms in ER operate on Populations of candidate controllers, initially selected for some distribution. Wherever, this population is then repeatedly modified according to fitness function. In the case of Genetic Algorithms, a common method in evolutionary computation, the population of candidate controllers is repeatedly grown according to crossover, mutation and other GA operators and then called according to the fitness function.

Implement Path Testing Using GA (Total Path)

Path testing is a structural testing method that finds every possible executable path from the source code of a program. The method ensures that every path through a program has been executed at least once. One of the major difficulties in the automation of software testing is automatic generation of adequate set of test data that satisfies the complete path coverage of a given program. Since it is impossible to cover all paths in software, the path testing method selects a subset of paths to execute and find test data to cover it. Many attempts were made to automate the test data generation process for path testing and suffered many limitations as the test data generation process is extensive and difficult process. The paper focuses on the Genetic Algorithm approach which is an adaptive heuristic search algorithm that premise on the evolutionary ideas of natural selection and genetic, to automate test data generation for a target path. This paper presents the utility and implementation of GA to automatically generate the test data to ensure the complete coverage of the target path.

4. STATE CHART DIAGRAM

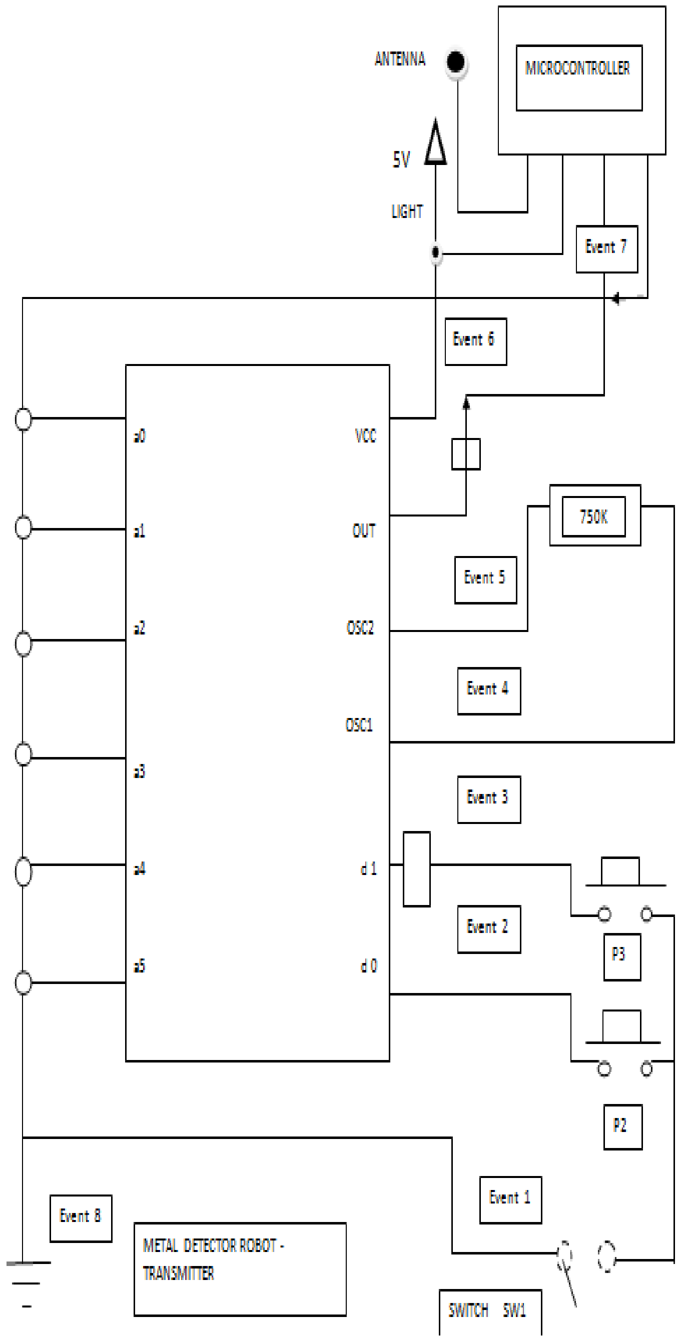

Fig 1: State-chart diagram

5. CONTROL FLOW GRAPH

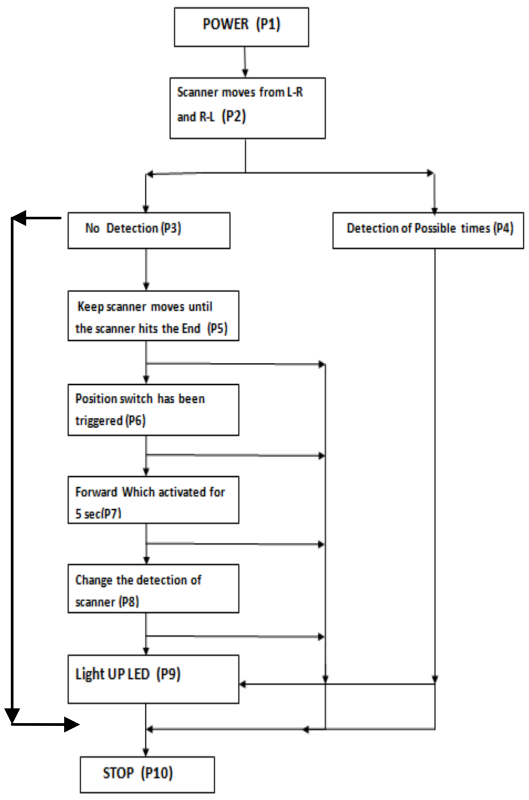
POWER (P1)
Scanner moves from L-R and R-L (P2)
No Detection (P3)
Detection of Possible times (P4)
Keep scanner moves until the scanner hits the End (P5)
Position switch has been triggered (P6)
Forward Which activated for 5 sec(P7)
Change the detection of scanner (P8)
Light UP LED (P9)
STOP (P10)

Fig 2. Control Flow Graph

6. METHODOLOGY

State chart diagram

1. Collect all Objects, method of scanner and detection.
2. For every conditional Detection, one mode two edges
3. For every possible time, there is detection or not which represented one node for each condition, one for true and other for false.
4. For every other detection there allot one node and one edge.
5. Connect the nodes and edges according to the order they occur.

7. PROPOSED METHOD

1. Draw the state chart diagram.
2. Convert state chart diagram to control flow graph using the above said algorithm.
3. Assign weights to the nodes as the parents weight is the weight of the node. If a node has multiple parents then add the weights of the parents to be the weight of the node.
4. Apply Genetic Algorithm to generate all paths between the source and destination with loops.
5. To calculate fitness value
    - For each path calculate the cost of the path, here the path's cost will be the sum of all the costs assigned to each link in the path
    - Apply the fitness functions as F(X)=X*X
    - Calculate the probability of the individual as
    - P (i)= F(x)/∑F(x)
6. To select the individuals from large initial population produce a new generation of solutions by picking from the existing pool of solutions with a preference for solutions which are better suited than others.
    - The probability range is divided into bins, sized according to the relative fitness of the solution which they represent.
    - By generating random values and seeing the bin where it fall into, to pick the individuals that will form the basis of the next generation.
7. To perform crossover, mate the first two strings together and the second two strings together and so on. For the first two pairs perform single point crossover on the fifth bit from left. For the next two pairs perform single point crossover on the third bit from left.
8. Mutate every fourth bit if random number generated is less than 0.2 to obtain the best path. 10. Reevaluate fitness of the new generation.
9. Repeat this process until the fitness value minimizes or all the paths have been covered or maximum number of generations is reached
10. Best Test path generated.
11. End.

8. CASE STUDY

The Genetic Algorithm (GA) is an optimization algorithm based on biological evolutionary system, mainly based on principal of "survival of fittest" proposed by Charles Darwin. It is a metal detector robot, which start from power on , scanner moves to detect metal until no metal is detected. To detect the metal, genetic algorithm proved that it is the best optimization technique to generate path and gives the best path for detection the metal in less money and efforts. With the help graph, generate all possible path, select path randomly and apply GA and its operator.

Randomly selected path

The Possible unique paths generated from the above graph are

Srcnode: P1=>P2=>P4=>P9=>P10: dstnode, cost= 35

Srcnode:P1=>P2=>P3=>P5=>P6=>P7P8=>P9=>P10:dstnode , cost=66

srcnodeP1=>P2=>P3=>P5=>P9=>P10: dstnode, cost=35

Srcnode P1=>P2=>P3=>P5=>P10: dstnode, cost=66

Number of all possible paths = 4

Table 1.

S.No.	Chromosomes	X	X*X	Probability	Associated Bin
1	00100011	35	1225	0.10763	0-0.10763
2	01000010	66	4356	0.38274	0.1-0.49037
3	00100110	35	1444	0.10763	0.4-0.61724
4	01000010	66	4356	0.38274	0.6-1

11381

Table 2

Random no.	Fall ino bin	Selection	Crossover Point	crossover	mutation
0.8247	4	01001010	4	01000010	01000010
0.9158	4	01001010	4	01000010	01000010
0.1260	2	01000010	2	01000010	01001010
0.9132	4	01001010	2	01000010	01000010

Table 3

s.no.	Chromosomes	X	X*X	probability	Associated bin
1	01000010	66	4356	0.23490	0-0.2349
2	01000010	66	4356	0. 23490	0.1-0.4698
3	01001010	74	5476	0.29529	0.4-0.76509
4	01000010	66	4356	0. 23490	0.76-1

18544

Table 4

Random no.	Fall ino bin	Selection	Crossover point	crossover	mutation
0.8004	4	01001010	3	01000010	01001010
0.1429	1	01001010	3	01000010	01000010
0.4232	2	01001010	2	01000010	01000010
0.7167	3	01000010	2	01001010	01001010

Table 5

s.no.	chromosomes	X	X*X	probability	Associated bin
1	01000010	66	4356	0.22152	0-0.22152
2	01001010	74	5476	0. 27847	0.2-0.49999
3	01000010	66	4356	0.22152	0.4-0.72151
4	010001010	74	5476	0. 27847	0.72-1

19664

Table 6

Random no.	Fall ino bin	Selection	Crossover point	crossover	mutation
0.7911	4	01001010	4	01001010	01001010
0.9695	4	01001010	4	01001010	01001010
0.6667	3	01000010	3	01000010	01000010
0.0329	1	01000010	3	01000010	01000010

Table 7

s.no.	chromosomes	X	X*X	probability	Associated bin
1	01001010	74	5476	0.26347	0-0.26347
2	01001010	74	5476	0. 26347	0.2-0.52694
3	01000010	66	4356	0.22152	0.5-0.73652
4	010001010	74	5476	0. 26347	0.73-1

20784

Table 8

Random no.	Fall ino bin	Selection	Crossover Point	crossover	mutation
0.3821	2	01001010	5	01001010	01001010
0.6524	3	01000010	5	01000010	01000010
0.1812	1	01001010	3	01000010	01000010
0.7030	3	01000010	3	01001010	01000010

Table 9

s.no.	Chromosomes	x	X*X	probability	Associated bin
1	01001010	74	5476	0.29529	0-0.29529
2	01000010	66	4356	0. 23492	0.2-0.53019
3	01000010	66	4356	0.23490	0.5-0.76509
4	01000010	66	4356	0. 23490	0.7-1

18544

9. CONCLUSION

In software testing, the generation of testing data is one of the key steps which have a great effect on the automation of software testing. The paper discusses the algorithm that depends on the principles of genetic algorithms to generate test data that provide good coverage in terms of the paths it tests or visits within the application. This paper presented the test case generation by means of UML state chart diagram and control flow graph using Genetic Algorithm from which best test cases can be optimized. The greatest merit of genetic algorithm in program testing is its simplicity. Genetic algorithms are often used for optimization problems in which the evolution of a population is a search for a satisfactory solution given a set of constraints. The proposed experimental sets. This method for test case generation inspires the developers to improve the design quality and to find multiple test cases ready for execution. In future, it is possible to build an automatic tool using this approach. This automatic tool will reduce cost of software development and improve quality of the software. Have used different combinations of the GA operator to find the test data for a target path in the CFG of a program under test.

10. REFERENCES

[1] A. Bouchachia, “An Immune Genetic Algorithm for Software Test Data Generation”, Seventh International Conference on Hybrid Intelligent Systems, 0-7695-2946-1/7 © 2007 IEEE. pp.84-89.

[2] X. Shen, Q. Wang, P. Wang, Bo Zhou, “Automatic Generation of Test Case based on GATS Algorithm”, 2007AA04Z148, supported by Nation 863 Project.

[3] P.R. Srivastava, T. Kim, “Application of Genetic Algorithm in Software Testing”, International Journal of Software Engineering and Its Applications, Vol. 3, No. 4, October 2009, pp.87-96.

[4] A. Rauf, S. Anwar, “Automated GUI Test Coverage Analysis using GA”, 2010 Seventh International Conference on Information Technology, 978-0-7695-3984-3/10 © 2010 IEEE, pp.1057-1062

[5] M. Harman, “Automated Test Data Generation using Search Based Software Engineering”, Second International Workshop on Automation of Software Test (AST'07) 0-7695-2971-2/07 $20.00 © 2007IEEE.

[6] M. A. Ahmed, I. Hermadi, “GAbased multiple paths test data generator”, Computers and Operations Research (2007).

[7] Ashalatha Nayak, Debasis Samanta: "Automatic Test Data Synthesis using UML Sequence Diagrams", in Journal of Object Technology, vol. 09, no. 2, March{April 2010, pp. 75{104,

[8] Li Bao-Lin, Li Zhi-shu, Li Qing, Chen Yan Hong ,” Test Case automate Generation from UML Sequence diagram and OCL Expression”, International Conference on Computational Intelligence and Security 2007, pp 1048-52.

[9] Monalisa Sarma Debasish Kundu Rajib Mall, “Automatic Test Case Generation from UML Sequence Diagrams”,15th International Conference on Advanced Computing and Communications 2007, pp 60-65.

[10] A. Abdurazik, and J. Offutt, Using UML Collaboration diagrams for static checking and test generation, in: Proceedings of the Third International Conference on the UML, Lecture Notes in Computer Science, Springer-Verlag GmbH, York, UK, vol. 939,2000,pp.383–395.