ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 4, Issue 2, February 2016

Test Case Generation from Activity Diagram Using Multiobjective Evolutionary Algorithm

Sukhjinder Kaur

Assistant Professor, Dept. of CSE, Continental Group of Institutes, Fatehgarh Sahib, Punjab, India

ABSTRACT: The software industry has become one of the world's key industries in recent decades. The tremendous amount of growth in software development industry has taken a pace and has become a driving force. It has grabbed the attention of researchers due to its subtle impact on world's economy and society. Software engineering deals with the design and development of high quality and reliable software. The overall objective in developing software is to provide high quality software without errors and failures. In order to produce high quality software which confirms to be the requirement specifications, it is necessary to test the software. Testing is required to make the software error free.This paper also highlights different techniques used for test case generation. Multi-objective formulations are realistic models for many complex engineering optimization problems. Customized genetic algorithms have been demonstrated to be particularly effective to determine excellent solutions to these problems. In many real-life problems, objectives under consideration conflict with each other, and optimizing a particular solution with respect to a single objective can result in unacceptable results with respect to the other objectives. A reasonable solution to a multi-objective problem is to investigate a set of solutions, each of which satisfies the objectives at an acceptable level without being dominated by any other solution. This paper describes a method using multi-objective evolutionary algorithm for the automatic generation of test cases.

KEYWORDS: Random;cyclomatic complexity; fitness factor; multi-objective genetic algorithm; test data.

I. INTRODUCTION

Software testing is an important phase during software development life cycle which is used to identify the defects in the system. It is an investigation conducted to provide stakeholders with information about the quality of the product or service under test .Software testing can also provide an objective, independent view of the software to allow the business to appreciate and understand the risks of software implementation. Test techniques include the process of executing a program or application with the intent of finding software bugs (errors or other defects).Software testing involves the execution of a software component or system component to evaluate one or more properties of interest. In general, these properties indicate the extent to which the component or system under test:

- Meets the requirements that guided its design and development
- Responds correctly to all kinds of inputs
- Performs its functions within an acceptable time
- Is sufficiently usable

Software testing can be conducted as soon as executable software exists. The overall approach to software development often determines when and how testing is conducted. For example, in a phased process, most testing occurs after system requirements have been defined and then implemented in testable programs. Software technology has seen great advances but has also witnessed some pretty major failures. Following are the examples of some major failures or errors in the software:

- Radiation dosage error happened in Panama City in 2000, where therapy planning software from US CompanyMultidata delivered different doses depending on the order in which data was entered. This resulted in massive overdoses for some patients, and at least five died.
- In 1996, a European Ariane 5 rocket was set to deliver a payload of satellites into Earth orbit, but problems with the software caused the launch rocket to veer off its path a mere 37 seconds after launch. More than $370 million were lost due to this error.

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 4, Issue 2, February 2016

- The Mars Climate Orbiter crashed in September 1999 because of a "silly mistake": wrong units in a program.
- In 1982 the Vancouver Stock Exchange instituted a new index initialized to a value of 1000.000. The cause was that the updated value was truncated rather than rounded.

A. Unified Modeling Language

The Unified Modeling Language (UML) is a standard visual modeling language intended to be used formodeling business and similar processes,analysis, design, and implementation of software-based systems.UML is a common language for business analysts, software architects and developers used to describe, specify, design, and document existing or new business processes, structure and behavior of artifacts of software systems.
UML can be applied to diverse application domains e.g., banking, finance, internet, aerospace, healthcare, etc. It can be used with all major object and component software development methods and for various implementations platforms.UML is a standard modeling language, not a software development process. UML Specification explained that process:

- provides guidance as to the order of a team's activities,
- specifies what artifacts should be developed,
- directs the tasks of individual developers and the team as a whole, and
- Offers criteria for monitoring and measuring a project's products and activities.

UML is intentionally process independent and could be applied in the context of different processes. Still, it is most suitable for use case driven, iterative and incremental development processes. An example of such process is Rational Unified Process (RUP).UML is not complete and it is not completely visual. Given some UML diagram, we can't be sure to understand depicted part or behavior of the system from the diagram alone. Some information could be intentionally omitted from the diagram, some information represented on the diagram could have different interpretations, and some concepts of UML have no graphical notation at all, so there is no way to depict those on diagrams. For example, semantics of multiplicity of actors and multiplicity of use cases on use case diagrams is not defined precisely in the UML specification and could mean either concurrent or successive usage of use cases.

Activity diagrams illustrate the dynamic nature of a system by modeling the flow of control from activity to activity. An activity represents an operation on some class in the system that results in a change in the state of the system. Typically, activity diagrams are used to model workflow or business processes and internal operation.

B. Test Case Generation

Test analysis and identifying test conditions gives us a generic idea for testing which covers quite a large range of possibilities. But when we come to make a test case we need to be very specific. A test case has a component that describes an input, action or event and an expected response, to determine if a feature of an application is working correctly. The basic objective of writing test cases is to validate the testing coverage of the application. Every organization has different testing processes and procedures. Manual Testing is important and irreplaceable- however automation is picking speed.

Automatic test case generation

Automation tool reduces the manual effort by using a scripting or programming language. Test automation costs are higher initially. There is the cost of the tool, and then there is a cost of the test automation resource and his/her training. But when scripts are ready, they can be executed hundreds of times repeatedly with the same accuracy and rather quickly. This also saves many hours of manual testing. So the cost gradually decreases, and ultimately it becomes the cost-effective method of regression testing.
Following are the situations, which cannot be tested manually:

1. Comparing two images pixel by pixel
2. Comparing two spreadsheets containing thousands of rows and columns.
3. Testing the application under the load of 100,000 users.
4. Performance Benchmarks.
5. Testing the application on different browsers and on different operating systems in parallel.

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 4, Issue 2, February 2016

Automated Unit Tests are written to test on code level. Bugs are identified in the functions, methods and routines written by developers. These resources have access to source code and they write unit tests to break the production code. **Automated Web Service / API Tests**. An Application Programming Interface makes it possible for software to talk to other software applications. Just like any other software, APIs need to be tested. **Automated GUI Tests** is the toughest form of automation because it involves testing of User interface of the application. It is tough because GUI's are highly subject to change. But this type of testing is also closest to what users will do with our application. The most popular GUI testing tools are QTP(UFT),Selenium, Test Complete and Microsoft Coded UI.

C. Multi-Objective Optimization

Multi-objective optimization also known as multi-objective programming is an area of multiple criteria decision making, that is concerned with mathematical optimization problems involving more than one objective function to be optimized simultaneously. Minimizing cost while maximizing comfort while buying a car, and maximizing performance whilst minimizing fuel consumption and emission of pollutants of a vehicle are examples of multi-objective optimization problems involving two and three objectives, respectively. For a nontrivial multi-objective optimization problem, there does not exist a single solution that simultaneously optimizes each objective. In that case, the objective functions are said to be conflicting, and there exists a (possibly infinite) number of Pareto optimal solutions.

Multi-objective optimization has also been increasingly employed in chemical engineering. In Fiandaca and Fraga, (2009),[11] the multi-objective generic algorithm (MOGA) was used to optimize the design of the pressure swing adsorption process. The design problem considered in that work involves the bi-objective maximization of nitrogen recovery and nitrogen purity. The results obtained in that work provided a good approximation of the Pareto frontier with acceptable trade-offs between the objectives. When decision making is emphasized, the objective of solving a multi-objective optimization problem is referred to supporting a decision maker in finding the most preferred Pareto optimal solution according to his/her subjective preferences. The underlying assumption is that one solution to the problem must be identified to be implemented in practice. Here, a human decision maker (DM) plays an important role. The DM is expected to be an expert in the problem domain.

II. RELATED WORK

It reviews the concept, existing algorithms and various techniques involved in automatic test case generation including model based testing and evolutionary techniques. This Section presents the literature surveyed on automatic test case generation and discusses different techniques for generating the test cases.

S. Yemul et al(2014) proposed approach to generate test case generation from UML behavior model and then, we optimize test coverage while minimizing time and cost. Model based testing approach for both test case generation and test case optimization for object-oriented softwares using UML diagrams. The proposed approach addresses the issue of redundancy, size of test cases and optimization challenges. Automation of test case design process can result in significant reductions in time and effort, and at the same time it can help in improving reliability of the software through increased test coverage. The proposed approach uses genetic algorithm from which best test cases can be optimized. Moreover our method for test case generation inspires the developers to improve the design quality.

V.Mary et al (2014) presented the test case generation of UML activity diagram. The four case studies Airport Departure Flow Management System (ADFMS), Automated teller Machine (ATM), Hospital Management system (HMS), and Edit Trend Properties (ETP) are used in experimenting to find the reduced set. The proposed algorithms were applied to cover all nodes, edges, and branches on the generated test suite. Greedy heuristic and the other evolutionary algorithms were applied to find the coverage criteria.

Luken et al (2014) presented a survey on multi-evolutionary algorithms for many objective problems. It highlighted various methods for multi-objective optimization as well as main findings in the field posing open research questions in this field. It highlighted various issues like influence of dimension on the problem hardness for MOEAs and visualization in MOEAs. Various methods discussed are scalarization based methods including aggregation and

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 4, Issue 2, February 2016

decomposition approach, dimensional reduction techniques, indicator based MOEAs and MOEAs based on space partitioning.

SaswatAnand et al (2013) presented an orchestrated survey on most prominent techniques for automatic test case generation of test cases. The techniques presented include structural testing with the help of symbolic execution, model-based testing, combinatorial testing, random testing, adaptive random testing and search-based testing. They briefly explained the basics ideas underlying the technique, the current state of art, discussion of the open research problems and a perspective of the future development in the approach. This paper ensured comprehensiveness and authoritativeness by giving an introductory, up-to-date and a brief overview of research in the field of automatic test case generation.

Mahesh Shirole et al (2013) focused on a survey on various UML behavioral model based test case generation. They focused on test case generation from the behavioral specification diagrams, namely sequence, state chart and activity diagrams. They classified various research approaches that are based on formal specifications, graph theoretic, heuristic testing, and direct UML specification processing. They also discussed the issues of test coverage associated with these approaches. In this paper, they analyzed different types of UML transition sequence based research for automating test case generation. A type of testing addressed by these behavioral models is functional testing. The techniques explored are based on design specifications or intermediate models of design specifications such as formal models, graphs, trees, and other hybrid types which are categorized as graph theoretic testing, heuristic testing, and direct UML specification processing.

S. Priyaet al (2013) presented a survey on different UML models for generating test cases. This paper highlighted the use of various UML diagrams for generating test cases. It included research work done on UML behavioral as well as structural diagrams. Apart from this, they reviewed combinational approach including combination of different diagrams and techniques for test case generation. It also focused on different tools for UML modeling like Umbrello, Astade, FUJABA, AgroUML, and Coral. Various open source drawing tools are: DIA, Violet, UMLet. The various commercially available modeling tools that support UML Diagrams are: Magic Draw, IBM Rational Rose, JUDE, SmartDraw etc.

III. PROPOSED OBJECTIVES

The objectives of the paper are as follows:

1. To propose and implement a method using multi-objective evolutionary algorithm for the automatic generation of test cases.
2. To compare the multi-objective evolutionary algorithm based proposed approach with existing single objective genetic algorithm based approach for generating test cases on the basis of time and fitness factor.

IV. RESEARCH PROBLEM FORMULATION

Test cases can be generated manually as well as automatically. Generating test cases manually consumes a lot of resources including time, money, man power etc. Manual testing becomes much more exhaustive if the software is too large and complex. Therefore automatic test case generation becomes a mandatory requirement to produce fast, efficient and reliable test cases resulting into good quality software with less effort, time and cost. For this, multiple techniques have been worked upon individually as well as using a combination of different techniques like search based testing, model based testing etc. UML has been adopted as a universal artifact for modeling the static and dynamic behavior of the system. Different types of diagrams like sequence diagrams, activity diagrams etc have been used to generate test cases automatically.

The work done till now is being limited to the application of genetic algorithms on different types of UML models. Enriching the research area and using the benefits of both model based testing and multi-objective evolutionary algorithm will give a new technique for automatic generation of test cases. In this approach, the application will be modeled into activity diagram and it will be converted into its corresponding tree structure through which the test cases would be derived. Then a multi objective approach will be applied to generate test cases. This approach will be further coupled with mutation analysis in order to check the effectiveness of the test cases.

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 4, Issue 2, February 2016

V. RESEARCH METHDOLOGY

The proposed algorithm is implemented with java program. XMI representation of activity diagram is taken as an input to the program and corresponding test cases are generated with the help of proposed algorithm. Figure 1 shows the Flowchart of Research Methodology.

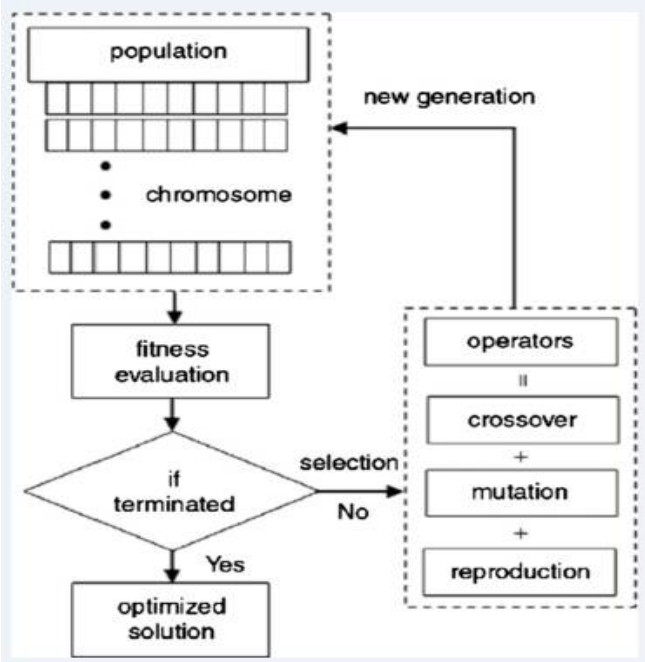

Fig 1. The Operation of Test Case Generation

An evolutionary algorithm (EA) is a stochastic optimization algorithm that simulates the process of natural evolution. Thus, an EA operates on a set of candidate solutions, which are subsequently modified by simplified implementations of the two basic principles of evolution: selection and variation. Selection represents the competition for resources among living beings. Some are better than others and more likely to survive and transmit their genetic information. A stochastic selection process simulates natural selection. Each solution is given a chance to reproduce a certain number of times, dependent on their quality, which is assessed by assigning a fitness value to each individual.

Algorithm for Proposed Approach

The sequence for generating the test cases from multi-objective evolutionary algorithm is as follows:

•The starting chromosome is generated from .xmi file obtained from ARGOUML.
•Quality score is calculated on the basis of average of time taken for generation of new population and space calculated from cyclomatic complexity.
•Selection and crossover methods are applied to generate new set of chromosomes.
•Mutation testing is applied to check the accuracy of test cases by analyzing the new population of binary trees.
•End

VI. RESULTS AND ANALYSIS

Results indicate that the approach is able to detect the seeded faults of mutation analysis process. The proposed technique for test case generation from multi objective algorithm is compared with single objective genetic algorithm on the basis of fitness factor and time and space. Results indicated that there is an improvement in reduction of time in

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 4, Issue 2, February 2016

case of multi objective genetic algorithm. The results of the proposed methodology for generating test cases from activity diagrams using multi-objective evolutionary algorithm are as follows:

A. Activity Diagram

Different types of tools exist for drawing UML diagrams. We have used ArgoUML tool to draw UML activity diagram.

B. Generation of Initial chromosomes Fitness function basically returns a number which quantifies how good a particular solution is at solving the problem. For example, in the case of the time table scheduler, a fitness function would be the number of clashes between lessons that are caused by a given schedule.

Fitness = (sub1 + sub2) – (sub1 - sub2)
Fitness = min (sub1, sub2)
Fitness = min (sub1, sub2, sub3,subn)

Quality factor is calculated by running the application at different time values at Time 1, Time 2 and Time 3. Average time is the average of Time1, Time 2 and Time 3.normalization is applied by dividing with 10.

Table 1: Quality Factor Calculation table

Test Case	Test Inputs	Time 1	Time 2	Time 3	Average Running Time	Time Factor	Space Factor	Quality Factor
1	2,3000,1,2000,100,5	810707	872156	781387	814750	6.23	7.2	7.22
2	1,2000,1,1000,101,4	514497	672937	679015	602150	6.37	7.1	6.40
3	1,1000,1,-1500,100,3	418462	406941	536642	463982	4.98	6.91	5.60
4	2,2000,1,-1500,101,5	541096	545108	558665	547260	5.81	7.5	6.90
5	1,2000,2,2000,100,5	914704	751669	868233	844735	7.40	7.90	6.81
6	1,2000,2,1500,101,4	894932	897558	879855	894582	7.58	7.13	7.90
7	1,3000,2,4000,100,2	677250	678335	490619	617868	5.43	8.31	6.71
8	1,1000,2,4000,101,5	741146	727775	844882	771268	7.10	8.10	7.45
9	1,2000,3,4	946049	1036674	1075139	1018254	9.90	7.61	8.05
10	1,3000,4,3,5	401438	527185	524753	481125	5.22	6.11	5.67

C. Implementation of MOEA

After the generation of trees for all possible operations and inputs obtained from XMI file, multi-objective Evolutionary algorithm is applied. A new set of population is generated after selection and crossover and for each sequence of events with respect to a given operation for which general tree structure is drawn. All the trees generated from new population are converted into binary trees. Binary trees are traversed using a traversal technique to generate a new set of test cases. Depth first search is used to traverse the binary trees. Validity of these test cases is checked, if the process terminates successfully, it is a valid test case, else invalid test case. Validation of test cases is shown in Table 2.

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 4, Issue 2, February 2016

Table 2: DFS validation table to check the validity of test case

S. No.	Test cases	Validation
1	2,3000,1,2000,100,5	Invalid
2	1,2000,1,1000,101,4	Valid
3	1,1000,1,-1500,100,3	Invalid
4	2,2000,1,-1500,101,5	Valid
5	1,2000,2,2000,100,5	Valid

The test cases are obtained from the binary trees by applying traversing which are presented in DFS validation table as shown above in Table 2 and then validity of each test case is checked depending on their completion of process/operation.

D. Mutation Analysis

Mutation analysis is the process of injecting the faults in the system and used to check the effectiveness of test cases which are generated from UML activity diagram. Below is the table which gives the description of faults found when faults are injected in the test cases of banking application. The graph of mutation analysis is given in Figure 4.

Mutation score = (Total number of faults found / Total number of faults injected) * 100

For bank Application, injected mutants are 27 and number of bugs detected are 23 are revealed from the test cases generated. Using the above formula, 87% score is achieved for the application which shows efficiency level of our approach. Fault coverage has been presented in Table 3 by using different operators.

Table 3: Fault coverage during mutation analysis

Fault Coverage Report		
Operator	Mutant given	Error found
Relational Operator	3	4
Data Value	14	11
Data name	4	4
Arguments	6	5
Total	27	23

Comparison between Genetic Algorithm and Multi-Objective Evolutionary Algorithm

The proposed approach of multi-objective Evolutionary algorithm is compared with single objective genetic algorithm for the generation of test cases from activity diagram and results indicate that multi-objective genetic algorithm produces better results as compared to single objective genetic algorithm. The proposed approach of multi-objective Evolutionary algorithm based testing approach is compared with single-objective genetic algorithm based approach on the basis of fitness factor, time and space. Table 4 shows the comparison of fitness factor of GA and MOEA.

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 4, Issue 2, February 2016

Table 4: Comparison of Fitness Factor

Test Case	Test Inputs	Quality Factor of GA	Average Running Time	Quality Factor of MOEA
1	2,3000,1,2000,100,5	6.4	0.27	2.82
2	1,2000,1,1000,101,4	7.4	0.17	3.2
3	1,1000,1,-1500,100,3	6.4	0.13	2.8
4	2,2000,1,-1500,101,5	6.4	0.18	2.79
5	1,2000,2,2000,100,5	8.1	0.23	4.3

Comparison of GA and MOEA on the basis of fitness factor

The proposed approach of generating test cases using MOEA is compared with GA on the basis of fitness factor. In genetic algorithm, only single objective is considered whereas in multi-objective Evolutionary algorithm, multiple objectives are considered namely time and space. The consideration of multiple objectives for test case generation is efficient as compared to single objective. Figure 2 represents the graphical representation of comparison of GA and MOEA fitness factor.

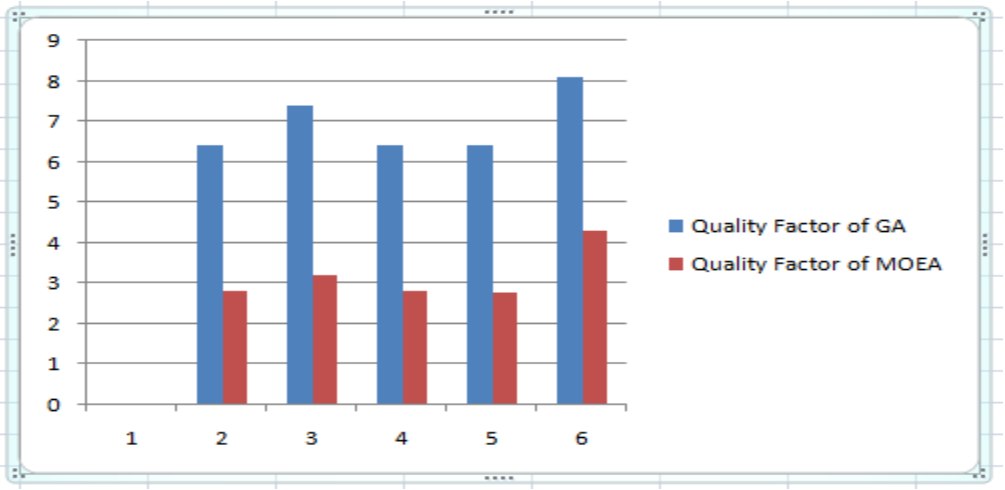

Fig 2. GA and MOEA Fitness Factor

The proposed technique gives the MOEA a significantly better choice of locating the global optimum solutions especially in case of problems with many constraints those results in a complicated search hypersurface. The varying fitness function of multi-objective Evolutionary algorithm outperforms than the fitness function of genetic algorithm while requiring the same computational time. Slight differences in the average time between the two algorithms are due to the stochastic nature of the GA execution.

Comparison of MOEA and GA on the basis of Time factor

The proposed approach of generating test cases using Multiobjective Evolutionary algorithm is compared with GA on the basis of time factor. In GA, time taken with respect to a particular test input is more as compared to time taken for multi-objective genetic algorithm. GA consumes more time as compared to Multiobjective Evolutionary algorithm. Table 5 shows the comparison of time for MOEA and GA respectively.

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 4, Issue 2, February 2016

Table 5: Time Factor of MOEA and GA

Test Case	Test Inputs	Average Running Time	Time Factor of MOEA	Time Factor of GA
1	2,3000,1,2000,100,5	854734	0.27	0.28
2	1,2000,1,1000,101,4	538017	0.17	0.19
3	1,1000,1,-1500,100,3	463082	0.13	0.15
4	2,2000,1,-1500,101,5	558262	0.18	0.18
5	1,2000,2,2000,100,5	731416	0.23	0.24

We provided a systematic comparison of multiobjective evolutionary approach to genetic approach using the chosen fitness functions. Each function involves a particular feature that is known to cause difficulty in the evolutionary optimization process, mainly in converging to the Pareto-optimal front (e.g., multimodality and deception). By investigating these different problem features separately, it is possible to predict the kind of problems to which a certain technique is or is not well suited. However, in contrast to what was suspected beforehand, the experimental results indicate the improvement in time factor as compared to genetic algorithm. Figure 3 represents the graphical representation of comparison of Multiobjective Evolutionary algorithm and GA time factor respectively.

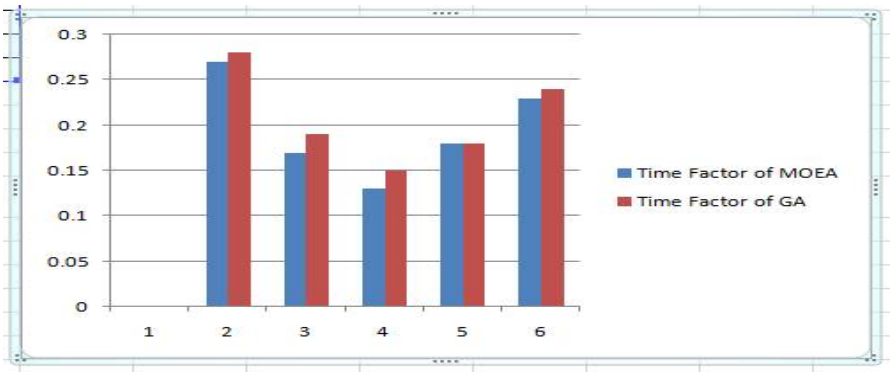

Fig. 3 Comparison of MOEA and GA Time Factor

VII. CONCLUSIONS AND FUTURE WORK

Testing is an important activity to be performed during the development of software since the software failures could prove to be highly expensive and more dangerous. Testing ensures the reliability of the software by making it error free. This work has focused on test case generation. UML has been adopted as a universal modeling language which is used to describe analysis and design specifications. It helps in visualizing the system with the help of different diagrams. UML diagrams have become sources for test case generation. The use of different evolutionary algorithms and UML based testing has further shown that test cases can be successfully generated covering single or multiple objectives like reduction in time, cost, increased fault count etc. This paper explored a multi-objective evolutionary technique for test case generation from UML Model and described a technique of optimization using multi objective genetic algorithm including multiple objectives. This paper outlined the way of test case generation for small projects. The technique can be expanded for more complex and bigger applications. In future, the approach can be employed to other UML diagrams like state diagrams, class diagrams etc.

REFERENCES

1. S. Yemul, K. Vhatkar and V. Bag, "Testing approach for automatic test case generation and Optimization using GA", International Journal of Emerging Trends & Technology in Computer Science, Vol. 3, pp. 2278-6856,2014.

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 4, Issue 2, February 2016

2. V.MarySumalatha and G.S.V.P.Raju, "Model Based Test Case Optimization of UML Diagrams using Evolutionary Algorithms", International Journal of Computer Science and Mobile Applications, Vol.2 Issue. 11, pp. 131-142:2321-8363, November- 2014.
3. C. Lucken, B. Baran and C. Brizuela, "A survey on multi-objective evolutionary algorithms for many-objective problems", ComputOptimAppl, Springer, 2014.
4. S. Anand et al., "An Orchestrated Survey on Automated Software Test Case Generation", Journal of Systems and Software, 2013.
5. M. Shirole and R., "UML Behavioral Model Based Test Case Generation: A Survey", ACM SIGSOFT Software Engineering Notes, vol. 38, 2013.
6. S. Priya and P. D. Sheba, "Test Case Generation from Uml Models – A Survey", International Journal of Emerging Technology and Advanced Engineering, vol.3, 2013.
7. R.K. Swain, V. Panthi and P.K. Behera, "Generation Of Test Cases Using Activity Diagram", International Journal of Computer Science and Informatics, vol. 3, Issue 2, 2013.
8. D. Liu,X. Wang, J. Wang, "Automatic Test Case Generation Based On Genetic Algorithm", Journal of Theoretical and Applied Information Technology, vol. 48, 2013.
9. A. Pachauri, G. Srivastava, "Automated test data generation for branch testing using genetic algorithm: An improved approach using branch ordering, memory and elitism", The Journal of Systems and Software, 2013.
10. D. Gong and Y. Zhang, "Generating test data for both path coverage and fault detection using genetic algorithms", Frontier Computer Science, vol. 7, pp. 822–837, 2013.
11. V. Panthi and D.P. Mohapatra, "Automatic Test Case Generation Using Sequence Diagram", International Journal of Applied Information Systems (IJAIS), vol. 2, May 2012.
12. J. Lee, S. Kang and D. Lee, "Survey On Software Testing Practices", IET Software, vol. 6, Issue 3, pp. 275–282, 2012.
13. PreetiGulia and R.S. Chillar, "A New Approach to Generate and Optimize Test Cases for UML State Diagram Using Genetic Algorithm", ACM SIGSOFT Software Engineering Notes, vol. 37, 2012.
14. S. Jiang, Y. Zhang and D. Yi, "Test Data Generation Approach for Basis Path Coverage", ACM SIGSOFT Software Engineering Notes, vol. 37, 2012.
15. A.V.K.Shanthi and G. Kumar, "Automated Test Cases Generation from UML Sequence Diagram", International Journal of Software and Computer Applications, vol. 41, Singapore, pp. 83-89, 2012.
16. J. Ferrer, F. Chicano and E. Alba, "Evolutionary algorithms for the multi-objective test data generation problem", Software Practiceand Experience, vol. 42, pp. 1331–1362, 2012.
17. H. Tahbildar and B. Kalita, "Automated Software Test Data Generation: Direction of Research", International Journal of Computer Science & Engineering Survey (IJCSES), vol. 2, 2011.
18. "UML Activity Diagram" available at URL:http://www.conceptdraw.com/examples/uml-activity-diagram.
19. "UML diagram" atURL:http://www.google.co.in.
20. "Introduction to ArgoUML" available at URL: http://argouml.tigris.org.