*I.J. Information Technology and Computer Science,* **2016, 8, 59-65**
Published Online August 2016 in MECS (http://www.mecs-press.org/)
DOI: 10.5815/ijitcs.2016.08.07

# Multi Objective Test Suite Reduction for GUI Based Software Using NSGA-II

**Neha Chaudhary**
Ph.D. Scholar, Gautam Buddha University, Greater Noida, India
E-mail: Neha.chaudhary@gmail.com

**O.P. Sangwan**
Guru Jambheshwer University of Science & Technology, Hisar, India
E-mail: sangwan_op@yahoo.co.in

***Abstract*—**Regression Testing is a performed to ensure modified code does not have any unintended side effect on the software. If regression testing is performed with retest-all method it will be very time consuming as testing activity. Therefore test suite reduction methods are used to reduce the size of original test suite. Objective of test suite reduction is to reduce those test cases which are redundant or less important in their fault revealing capability. Test suite reduction can only be used when time is critical to run all test cases and selective testing can only be done. Various methods exist in the literature related to test suite reduction of traditional software. Most of the methods are based of single objective optimization. In case of multi objective optimization of test suite, usually researchers assign different weight values to different objectives and combine them as single objective. However in test suite reduction multiple Pareto-optimal solutions are present, it is difficult to select one test case over other. Since GUI based software is our concern there exist very few reduction techniques and none of them consider multiple objective based reduction. In this work we propose a new test suite reduction technique based on two objectives, event weight and number of faults identified by test case. We evaluated our results for 2 different applications and we achieved 20% reduction in test suite size for both applications. In Terp Paint 3.0 application compromise 15.6% fault revealing capability and for Notepad 11.1% fault revealing capability is reduced.

***Index Terms*—**Test Suite Reduction, NSGA II, Multi Objective Optimization, Pareto-optimal solution.

## I. Introduction

The widespread recognition of the usefulness of graphical user interface (GUIs) has established their importance as critical components of today's software. Testing of GUIs systems is more difficult due to the following reasons: The event driven nature of GUIs, unsolicited events, many ways in/ many ways out, and the infinite input domain problems make it likely that the programmer has introduced errors because he could not test every path [6]. Regression testing means rerunning test cases from existing test suites to build confidence that software changes have no unintended side-effects. The ideal process for regression testing is to create a wide test suite and run it after each and every modification [7]. Regression testing is also a critical problem with GUI's. This is because the GUI may modify significantly across versions of the application, even though the underlying application may not. A small modification in GUI may cause many of test cases to become useless. When we do regression testing huge number of test cases becomes unusable for different version of application under test. Rerunning all test cases again will be time consuming. So we require test suite reduction technique for GUI based software. There are very few existing techniques for test suite reduction of GUI based software and they are based on single objective. In this work we propose Multi-objective test suite reduction technique for GUI based software.

One of the objectives of multi objective test suite reduction is to find as many Pareto-optimal test cases as possible. This discards the requirement of assigning weight values to multiple objectives converting them in a single one. Evolutionary algorithms consider all non dominated solution in a population as similar and provide a diverse set of multiple non dominated solutions [8]. That is why EA is a preferable choice for test suite reduction it will eliminate only dominated solutions.

Pareto-optimal solutions

Pareto optimal solution exists when there is a need to optimize multiple conflicting objectives, there is trade off between one or more conflicting objectives and the relative importance of these objectives is not known.

Therefore our objective of test suite reduction of GUI based software is

- To find test cases which are as close as possible to the Pareto-optimal front
- To identify test cases as diverse as possible to cover complete test suite

A multi objective test suite minimization problem can be formulated as:

Given a test suite T for GUI based application with Events E = $\{e_1,e_2,e_3,\ldots e_n\}$, where E is the set of events.

 

Find the minimal test suite T' such that T' is a Pareto-Optimal set which satisfy a given measure. (maximum weight based event coverage and consider set of fault identified by individual test cases)

The organization of this paper is as follows: Section II discuss about previous work done by researchers. Section III demonstrates Multi Objective Test Suite Reduction and problem formulation. Test suite reduction using NSGA II is discussed in Section IV. This section also comprises experimental results. Section V comprises of threats to validity. Finally, section VI contains conclusion and future work.

## II. Related Work

The objective of test suite minimization is to reduce the number of test cases in a given test suite which satisfy the given criteria. Many greedy algorithms are used to solve test suite reduction problems [25, 26, 27]. Harrold et al. proposed a heuristic based algorithm known as HGS for test suite reduction. That algorithm tries to minimize test suite based on program requirements.

The concept of HGS algorithm was further generalized by Von Ronne. In this work a concept of hitting factor was introduced, based on this factor every requirement could be satisfied multiple times.

To identify near optimal solutions for test suite reduction problem Chen & Lau apply divide and-conquer techniques.

The next frequently used test suite reduction techniques are based on evolutionary computation [30, 31].

The problem of test suite reduction can be formulated as NP-Hard problem [18], minimum set cover problem. For test suite minimization many heuristics are suggested in literature [20, 19]. There are few studies that report no impact of test suite reduction[21,22] and some studies shows negative impact of test suite reduction because reduce test suite will compromise the fault revealing capability of test suite [13]. That may be due to reduction criterion which is simply taken as structural coverage. Although, Rothermel et al. in their paper reveal that the fault-detection capabilities of test suites may be severely compromised by test-suite reduction. They further analyze the cost and benefit of test suite reduction. Authors specify that characteristics of programs and faults will be important parameters for test suite reduction [13].

In further research some more techniques are proposed and they have some sophisticated reduction criteria and consider program characteristics. Scott McMaster and Atif Memon present a reduction technique based on the call-stack coverage criterion. Significance of this technique is due to the context provided by call stack, which is valuable in test suite reduction [9]. Preethi Harris and Nedunchezhian Raju in their work reduce the size of the test suite based on two metrics Size and requirement coverage [10]. Rajiv Gupta and Mary Lou Soffa, presents a test suite reduction technique based on data flow testing methodology. They select a reduce test suite which provide same coverage as entire test suite by removing redundant and obsolete test cases[11]. A static analysis approach based on program slicing is proposed by Stephan Arlt et al. [12]. This approach reduces the size of test suite by removing redundant event sequences from test cases.

Previous work has considered test suite reduction problem as single objective optimization. From the last few years researchers have introduced the concept of Pareto efficiency for test suite reduction and consider multiple objectives like code coverage, past fault-detection history and execution cost [16] for reduction. Shin Yoo et al. developed a search based optimisation approach for multi objective regression test optimisation for graphics cards. Authors in the paper introduce the concept of parallel test suite minimization along with the concept of scalability [15]. Saeed Parsa and Alireza Khalilian in their paper consider test suite reduction as multi objective problem where first objective is fault detection capability and it has to be maximized. The second objective is number of test cases which should be minimized. They used greedy algorithm to solve this optimization problem.

There are very few researches which focus on test suite reduction of GUI based applications. In one of the approach reduction is based on call stack based coverage. In another research done by Wei Sun et al. a multi-objective algorithm is proposed for test cases prioritization for GUI applications. They consider statement coverage event coverage for prioritization criteria.

## III. Multi Objective Test Suite Reduction

Most evolutionary multi objective optimization algorithms require us to find best non dominated front in the population and in our approach best non dominated front is reduced set of test cases and reduction is based on following two objectives

1) Weight of test case
2) Number of faults identified

In our previous work we have generated a formula for calculating weight of each test case that is based on weight value of events and event coverage.

In this work event classification is considered where events are classified according to their fault revealing capability and they are assigned a weight value (events classification and their weight value is shown in table 1).

Table 1. Event weight assignment [1]

| Event type | WVs |
|---|---|
| Restricted-focus event | 5 |
| System-interaction event | 4 |
| Termination event | 3 |
| Menu-open event | 2 |
| Unrestricted-focus event | 1 |

 

Then weight of each event is added and multiplied with the coverage of test case.

Finally coverage is computed by calculating (adding) number of events in the test case divided by total number of events in the application [2].

Weight of each test case is calculated according to the formula given in (1):

$$W_{TC} = n / Tn * \sum_{j=1}^{n} Wj \quad (1)$$

Where $W_{TC}$ is Weight of test case, Wj is the $j^{th}$ event weight, n is the number of events in test case and Tn is the total number of events in AUT.

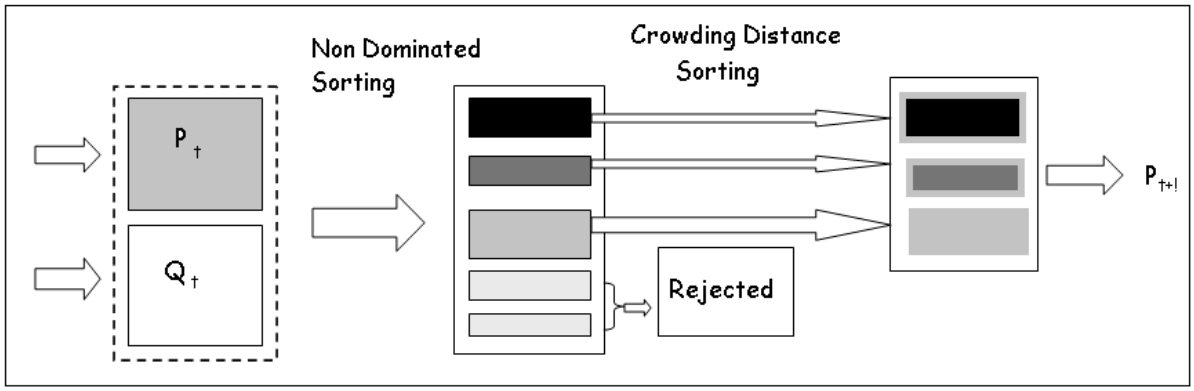


Fig.1. NSGA II procedure [8]

## IV. Test Suite Reduction Using NSGA II

NSGA-II uses an explicit diversity-preserving mechanism. In order to sort a population of size N according to the level of non-domination, each solution must be compared with every other solution in the population to find if it is dominated. NSGA II starts with the parent population $P_t$ and offspring population $Q_t$.. Procedure for algorithm is specified in fig. 1.

Various steps of algorithm are specified as follows:

1) Generation of Initial Set of test suite: In our application we have taken test cases of 'Terp Paint 3.0' from Event Driven Software Lab. To implement the algorithm we have randomly selected few test cases from artefacts. Test cases are represented in binary string format according to the number of fault they reveal and number of event coverage. For example test case T1 reveal fault 1 and event coverage 6 and T2 reveal fault 2 and coverage 14. Length of test case is 5 where first 5 bit represents number of faults covered by test case and last five bit represents number of events covered by test case.

Table 2. Binary representation of test cases

| Test Case | Number of faults | | | | | Event Coverage | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| T1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 0 |
| T2 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0 |
| T3 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 1 |
| T4 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |
| T5 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 0 |
| T6 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 0 |
| T7 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 |

Table 3. Fault and Weight value of test cases

| Test Case | Number of fault | Weight of test case |
|---|---|---|
| T1 | 1 | 1.61 |
| T2 | 2 | 10.04 |
| T3 | 4 | 10.04 |
| T4 | 3 | 3.61 |
| T5 | 4 | 3.61 |
| T6 | 7 | 3.61 |
| T7 | 6 | 6.43 |

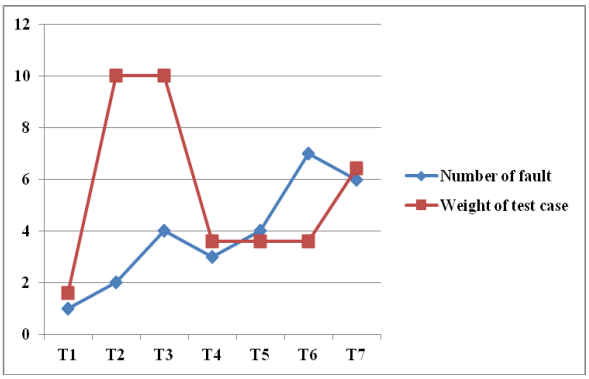


Fig.2. Line Chart for Number of Faults and Weight of Test Case

After generation of initial population their values are calculated for both objectives. Weight of test case is calculated according to (1) and the number of faults is identified from initial testing. These values are specified in table 3:

Fig 2 shows line chart for number of faults and weight of test cases it is clearly visible from the figure that these two are conflicting objectives test cases can not be

 

selected based on any one objective.

2) Perform Non-dominated sorting on initial test suite: In our example we have used Kung et al.'s efficient method for sorting. This method is most computationally efficient method.

Since both objectives represent maximization for test suite reduction. We need to combine parent population and offspring population & then we need to perform non-dominated sorting and identify different fronts.

For the given example we require offspring population also but for the example purpose we consider only parent population. Sort the test cases according to descending order of importance of Event Weight. Now we call set of test cases as T'

T'= {T2,T3,T7,T4,T5,T6,T1}

Next we have performed non dominated sorting on T' and we obtain following non dominated fronts:

F1 = {T3, T7, T6}
F2 = {T2,T5}
F3 = {T4}
F4 = {T1}

In this example total number of test cases is N but in the algorithm total number of test cases will be 2N and we need to identify N test cases from the initial fronts.

3) Calculate the crowding distance: Once the sorting is complete, crowding distance is assigned to each test case in all fronts. Crowding distance comparison does not matter in different fronts. Test cases are selected based on rank and crowding distance [4]. Crowding distance is used to select test cases from the same front. Test cases in the boundary are assigned infinity distance so these test cases are always selected. Crowding distance is computed according to (2).

$$F(d_k) = F(d_k) + \frac{F(k+1).m - F(k-1).m}{f_m^{\max} - f_m^{\min}} \quad (2)$$

F (k).m is the value of the $m^{th}$ objective function of the $k^{th}$ individual in F.

This step will return solutions which are diverse in the solution space. One selected test will be less crowded compare to other solutions.

4) Perform crowded tournament selection, crossover and mutation:

These operations will be performed to create offspring population of test cases $Q_{t+1}$ from $P_{t+1}$

A crowded tournament selection operator is used to select test cases, where a test case Ti will win the tournament if 'i' has a better rank or they have the same rank but 'i' has better crowding distance then other solution. We performed other tournaments to obtain the mating pool and then these test cases are mated pair wise and mutation is performed to generate next offspring population.

We have implemented NSGA II using MATLAB 7.10.0(R2010 a). For test suite reduction we have used two objective functions, weight of test case and number of faults identified by test case. Both functions have to be maximizing for reduction purpose.

## V. Evaluation of Algorithm

For the evaluation of test suite reduction methods, we have considered two applications i.e. Terp Paint 3.0 and Notepad. For both applications, when we run NSGA II algorithm we get results as shown in table 4. The results of the optimization shown in table 4 containing both objective function values and the value of the Number of Events and faults. We have evaluated test cases of both applications corresponding to given values. All test cases near Pareto optimal front are selected for regression testing purpose.

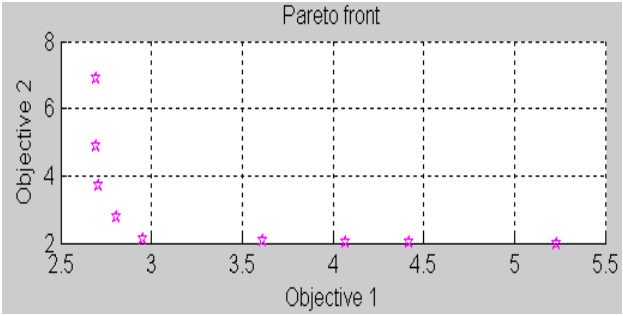


Fig.3. Pareto front for Test suite minimization

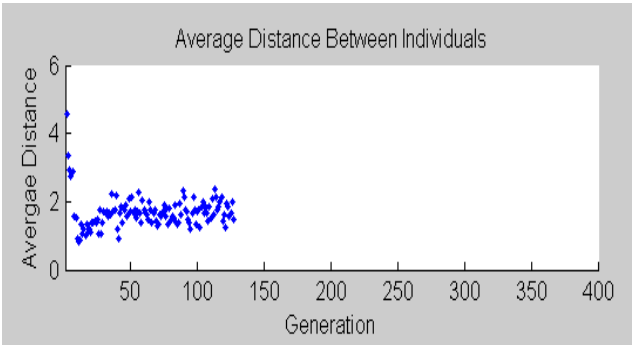


Fig.4. Average Distance between Individuals

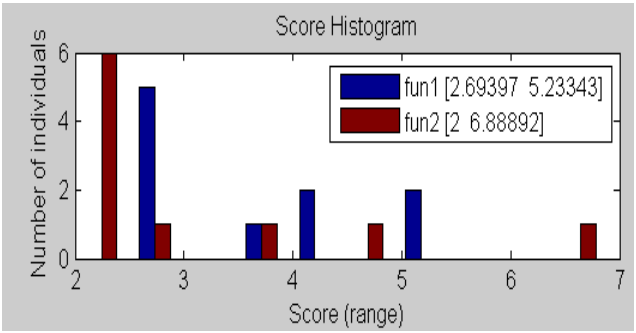


Fig.5. Score Histogram for two objectives

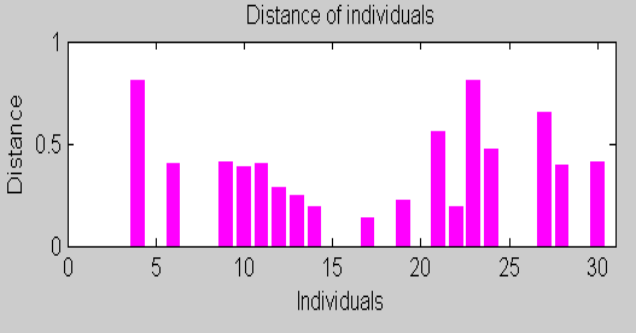


Fig.6. Distance between individuals

Fig. 3 represents two competing objectives. The tradeoff between these two objectives, weight of test case and number of faults is plotted in objective function space.

Fig. 4 to fig. 6 plots various aspects of the multi objective genetic algorithm its execution corresponding to different generations. In Table 4, Weight of test case and number of faults represents both objective functions. Number of Events and Faults represents input values.

Table 4. Optimal values for test suite reduction

| Test Case | Objectives | | Inputs | |
|---|---|---|---|---|
| | Weight of test case | Number of Faults | Number of Events | Faults |
| 10 | 13.10326 | 3.092452 | 11.1289 | 2.092452 |
| 1 | 11.24816 | 3.093185 | 11.12875 | 2.093185 |
| 2 | 9.266539 | 3.096041 | 9.592378 | 2.096041 |
| 8 | 4.454116 | 3.09611 | 6.378983 | 2.09611 |
| 5 | 3.588967 | 3.171592 | 5.638344 | 2.171592 |
| 11 | 2.862798 | 3.660142 | 5.382481 | 2.660142 |
| 3 | 2.817113 | 4.956395 | 5.335696 | 3.956395 |
| 4 | 2.724817 | 6.548348 | 5.240018 | 5.548348 |
| 6 | 2.724781 | 8.548256 | 5.23998 | 5.548256 |
| 9 | 2.724779 | 9.548562 | 5.239978 | 5.548562 |
| 7 | 2.724602 | 10.54852 | 5.239793 | 5.548515 |

To calculate the test suite size reduction and fault detection capability loss we have used (3) and (4) respectively:

$$\%\text{Test suite size reduction} = \frac{|T| - |T_{RED}|}{|T|} * 100 \quad (3)$$

$$\%\text{Fault detection loss} = \frac{|F| - |F_{RED}|}{|F|} * 100 \quad (4)$$

We have evaluated NSGA II using Application Under Test (AUT), Notepad and Terp Paint 3.0[3].

Table 5. Size Reduction and Fault Detection loss for Application Under Test

| AUT | % Size Reduction | % Fault Detection loss |
|---|---|---|
| Terp Paint 3.0 | 20 | 84 |
| Notepad | 20 | 88.8 |

As shown in table 5, we consider small size of test suite for Terp Paint 3.0 Application, it gave 20% reduction in test suite size and its fault revealing capability is reduced to 84.4%.

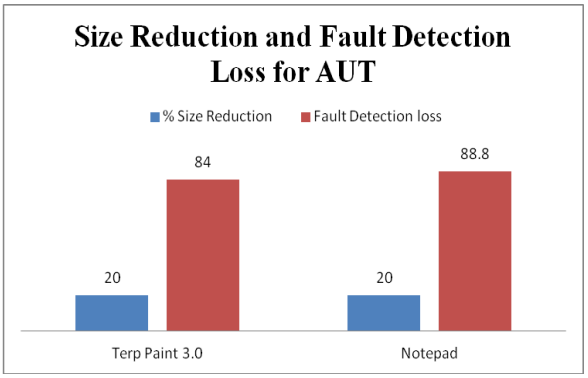


Fig.7. Bar chart for Size Reduction and Fault Detection loss for Application Under Test

In another example of notepad test suite size reduction is 80% and fault revealing capability is reduced to 88.8%. Fig. 7 represents bar chart for application under test considering reduction in test suite size and fault detection loss.

## VI. Threats to Validity

Threats to validity consider all aspects that may affect ability to generalize results in other situations. First threat considers that validation of our results is done using test cases generated for 2 applications, first is Terp Paint 3.0 and another application is Notepad. Test cases for Terp Paint are generated using Guitar tool which generates test cases by creating all possible combination of events by ripping the application. For notepad application we have generated test cases using HP-QTP version 11 [5]. Further experiments should be done with bigger size of test suites. There may be different cost associated with every test case execution and this is another threat to validity but we have considered uniform cost of execution in our research.

## VII. Conclusion and Future Work

We have implemented NSGA II algorithm for test suite reduction. When we executed algorithm for two examples, we are able to achieve reduction in test suite size. From the analysis of results obtained from two applications this is vibrant that whenever test suite size is reduced fault revealing capability will be compromised. In case of time constraint we should reduce number of test cases other than redundant ones because fault revealing capability of test suite will be reduced. As we infer from the literature that there is no existing technique for test suite reduction of GUI based software which considers multiple objectives. This is a novel idea for future research.

 

## REFERENCES


[1] Memon Atif, Lou Soffa Mary, E. Pollock Martha, "Coverage criteria for GUI testing", in the proceeding of 21st International conference on software engineering, ACM press, pp 257-266, 1999.

[2] Neha Chaudhary, O.P. Sangwan, Richa Arora," Event-Coverage and Weight based Method for Test Suite Prioritization", I.J. Information Technology and Computer Science, Vol. 12, pp. 61-66, 2014.

[3] Reference: Terp Paint 3.0 Fault matrix http://www.cs.umd.edu/~atif/Benchmarks/UMD2007a.html

[4] Kalyanmoy Deb and Deb Kalyanmoy, "*Multi-Objective Optimization Using Evolutionary Algorithms*" John Wiley & Sons, Inc., New York, NY, USA, 2001.

[5] Kaur and Kumari, HP QuickTest Professional version 11. 2010. HP – QTP version 11, Comparative study of Automated Testing Tools: Test Complete and QuickTest Pro, Punjab University, 2011.

[6] Paul Gerrard, "Testing GUI Applications", EuroSTAR, Edinburgh UK, 1997

[7] A. M. Atif, "Automatically Repairing Event Sequence-Based GUI Test Suites for Regression Testing", ACM Transaction on Software Engineering and Method. Volume 18, Issue 2, Nov. 2008.

[8] Kalyanmoy Deb, "Multi-Objective Optimization Using Evolutionary Algorithms", Wiley India Private Limited, ISBN-10: 8126528044.

[9] McMaster, S.; Memon, A.M., "Call-Stack Coverage for GUI Test Suite Reduction," *Software Engineering, IEEE Transactions on* Software Engineering, vol.34, no.1, pp.99-115, Jan.-Feb. 2008

[10] Preethi Harris and Nedunchezhian Raju," A Greedy Approach for Coverage-Based Test Suite", The International Arab Journal of Information Technology, Vol. 12, No.1, PP. 17-23, January 2015.

[11] M. Jean Harrold, Rajiv Gupta, and Mary Lou Soffa. 1993. A methodology for controlling the size of a test suite. *ACM Trans. Softw. Eng. Methodol.* 2, pp. 270-285, July 1993.

[12] Arlt, Stephan and Podelski, Andreas and Wehrle, Martin, "Reducing GUI Test Suites via Program Slicing", Proceedings of the 2014 International Symposium on Software Testing and Analysis, 2014, pp.270-281, 2014.

[13] G. Rothermel, M. J. Harrold, J. von Ronne, and C. Hong, "Empirical studies of test-suite reduction", Journal of Software Testing, Verification, and Reliability, Vol 12, no. 4, December, 2002.

[14] G. Rothermel, R. Untch, C. Chu, and M. J. Harrold, "Test case prioritization", IEEE Transactions on Software Engineering, vol. 27, no. 10, pp. 929-948, October, 2001.

[15] Shin Yoo, Mark Harman, and Shmuel Ur, "Highly scalable multi objective test suite minimisation using graphics cards", Proceedings of the Third international conference on Search based software engineering, pp. 219-236, 2011.

[16] Shin Yoo and Mark Harman, "Pareto efficient multi-objective test case selection", In Proceedings of the 2007 international symposium on Software testing and analysis (ISSTA '07), ACM, New York, NY, USA, pp. 140-150, 2007.

[17] Saeed Parsa and Alireza Khalilian," On the Optimization Approach towards Test Suite Minimization", International Journal of Software Engineering and Its Applications Vol. 4, No. 1, January 2010.

[18] M. R. Garey and D. S. Johnson, "Computers and Intractability: A guide to the theory of NP-Completeness", New York, NY: W. H. Freeman and Company, 1979.

[19] J. O_utt, J. Pan, and J. Voas, "Procedures for reducing the size of coverage-based test sets," in Proceedings of the 12th International Conference on Testing Computer Software. ACM Press, pp. 111-123, June 1995.

[20] T. Y. Chen and M. F. Lau, "Dividing strategies for the optimization of a test suite," Information Processing Letters, vol. 60, no. 3, pp. 135-141, 1996.

[21] W. E. Wong, J. R. Horgan, S. London, and A. P. Mathur, "Effect of test set minimization on fault detection effectiveness," Software Practice and Experience, vol. 28, no. 4, pp. 347-369, April 1998.

[22] W. E. Wong, J. R. Horgan, A. P. Mathur, and A. Pasquini, "Test set size minimization and fault detection effectiveness: A case study in a space application," The Journal of Systems and Software, vol. 48, no. 2, pp. 79-89, October 1999.

[23] G. Rothermel, S. Elbaum, A. Malishevsky, P. Kallakuri, and B. Davia, "The impact of test suite granularity on the cost-effectiveness of regression testing," in Proceedings of the 24th International Conference on Software Engineering (ICSE 2002). ACM Press, pp. 130-140, May 2002.

[24] S. Yoo and M. Harman, "Pareto efficient multi-objective test case selection," in Proceedings of International Symposium on Software Testing and Analysis. ACM Press, pp. 140-150, July 2007.

[25] M. Jean Harrold, Rajiv Gupta, and Mary Lou Soffa, "A methodology for controlling the size of a test suite" ACM Trans. Softw. Eng. Methodol. 2, 270-285, 1993.

[26] Jun-Wei Lin and Chin-Yu Huang, "Analysis of test suite reduction with enhanced tie-breaking techniques", Inf. Softw. Technol. 51, 679-690, 2009.

[27] Saeed Parsa and Alireza Khalilian," On the Optimization Approach towards Test Suite Minimization", International Journal of Software Engineering and Its Applications, Vol. 4, No. 1, January 2010.

[28] J. von Ronne, "Test Suite Minimization: An Empirical Investigation," university honors college thesis, Oregon State Univ., June 1999.

[29] T. Y. Chen and M. F. Lau, "Dividing strategies for the optimization of a test suite", Information Processing Letters, vol. 60(3), pp.135-141, March 1996.

[30] Xue-ying MA, Zhen-feng He, Bin-kui Sheng, Cheng-qing Ye, "A genetic algorithm for test-suite reduction," in Systems, Man and Cybernetics, 2005 IEEE International Conference on , vol.1, no., pp.133-139 Vol. 1, 10-12 Oct. 2005

[31] Yi-kun ZHANG, Ji -ceng LIU, Ying-an CUI, Xinhong EI, Ming-hui ZHANG, An Improved Quantum Genetic Algorithm for Test Suite Reduction, IEEE International Conference on Computer Science and Automation Engineering (CSAE), 2011.


### Authors' Profiles

**Neha Chaudhary** holds a Masters of Technology and a Bachelor of Engineering degree in Computer Science & Engineering. She is pursuing her Ph.D from Gautam Buddha University. Her research interest includes Web Testing, Software Testing, GUI Testing and metrics development. She has

many publications in international journals and conferences to her credit.

**Dr. Om Prakash Sangwan** received his PhD in Computer Science & Engineering and Master of Technology (M.Tech) degree in Computer Science & Engineering with distinction in Research Work from Guru Jambheshwar University of Science & Technology, Hisar, Haryana. He is also CISCO Certified Network Associate (CCNA) and CISCO Certified Academic Instructor (CCAI). His area of research is Software Engineering focusing on Planning, Designing, Testing, Metrics and application of Neural Networks, Fuzzy Logic and Neuro-Fuzzy. He has numbers of publications in International / National Journals and Conferences. He is presently (on EOL from Department of Computer Science & Engineering, School of Information & Communication Technology, Gautam Buddha University, Greater Noida, Uttar Pradesh) working as Associate Professor with Department of Computer Science & Engineering, Guru Jambheshwer University of Science & Technology, Hisar, Haryana,. Before joining the current assignment Dr. Sangwan has worked as Dy. Director with Amity Resource Centre for Information Technology (ARCIT), and LMC & Head, CISCO Regional Networking Academy, Amity Institute of Information Technology, Amity University, Uttar Pradesh. He is also Member of Computer Science Teacher Association (CSTA), New York, USA, International Association of Engineers (IAENG), Hong Kong, IACSIT (International Association of Computer Science and Information Technology, USA, professional member Association of Computing Machinery, USA, IEEE, and Life member, Computer Society of India, India. He has also published a book on Soft Computing Techniques in Software Engineering co-authored by Prof. Yogesh Singh, Hon'ble Vice Chancellor, M.S. University, Baroda, Gujarat.

**How to cite this paper:** Neha Chaudhary, O.P. Sangwan, "Multi Objective Test Suite Reduction for GUI Based Software Using NSGA-II", International Journal of Information Technology and Computer Science (IJITCS), Vol.8, No.8, pp.59-65, 2016. DOI: 10.5815/ijitcs.2016.08.07