ENHANCED NON-DOMINATED SORTING GENETIC ALGORITHM FOR TEST CASE OPTIMIZATION

IZWAN BIN MOHD ISMAIL

A dissertation submitted in fulfilment of the
requirements for the award of the degree of
Master of Computer Science

Faculty of Computing
Universiti Teknologi Malaysia

JUNE 2018

ACKNOWLEDGEMENT

First of all, I would like to express my gratitude to my dissertation supervisor, Assoc. Prof. Dr. Wan Mohd Nasir Wan Kadir for guidance and help from him for me to complete my project. His opinion and suggestion is highly appreciated. Next, I also would like to dedicate my appreciation to my parents for their pray and support to me in my effort for completing this dissertation. I also would like to thank my friends that giving opinion and helping me accomplish this dissertation. Finally, I would like to thank all lecturers in Faculty of Computing. They have taught me a valuable lesson that I used for completing this dissertation. Thanks to all.

ABSTRACT

Due to inevitable software changes, regression testing has become a crucial phase in software development process. Many software testers and researchers agreed that regression testing process consumes more time and cost during software development. Test case optimization has become one of the best solutions to overcome problems in regression testing. Test case optimization is focusing on reducing number of test cases in the test suite that may reduce the overall testing time, cost and effort of software testers. It considers multiple objectives and provides several numbers of optimal solution based on objectives of the testing. Therefore, this research aims at developing an alternative solution of test case optimization technique using NSGA II with fitness scaling as an additional function. Fitness scaling function is applied in NSGA II to eliminate pre-mature convergence among set of solution in the evolution of offspring in NSGA II which may produce more efficient fitness value. This research focuses on regression testing optimization by implementing weight of test cases and fault detection rate per test case as its objective function for optimization purposes. The proposed technique is applied to the GUI-based testing case study. The result shows that Pareto front produced by enhanced NSGA II give more wider set of solution that contains more alternatives and provide better trade-off among solutions. The evaluation shows that enhanced NSGA II perform better compared to conventional NSGA II by increasing the percentage of the reduced test cases with 25% and yield lower fault detection loss with 1.64% which indicating that set of reduced test cases using enhanced NSGA II is able to maintain the fault detection capability in the system under test.

ABSTRAK

Oleh kerana perubahan perisian yang tidak dapat dielakkan, ujian regresi telah menjadi fasa penting dalam proses pembangunan perisian. Banyak penguji perisian dan penyelidik bersetuju bahawa proses ujian regresi menggunakan lebih banyak masa dan kos semasa pembangunan perisian. Pengoptimuman kes ujian telah menjadi salah satu penyelesaian terbaik untuk mengatasi masalah dalam ujian regresi. Pengoptimuman kes ujian menumpukan kepada pengurangan bilangan kes ujian dalam satu ujian yang boleh mengurangkan masa ujian keseluruhan, kos dan usaha penguji perisian. Ia mengambil kira pelbagai objektif dan menyediakan beberapa penyelesaian yang optimum berdasarkan objektif ujian. Oleh itu, kajian ini bertujuan untuk membangunkan penyelesaian pengoptimuman kes ujian alternatif menggunakan NSGA II dengan skala kecergasan sebagai fungsi tambahan. Fungsi skala kecergasan digunakan dalam NSGA II untuk menghapuskan penumpuan pra-matang di kalangan set penyelesaian dalam evolusi keturunan yang dapat menghasilkan nilai kecergasan yang lebih cekap. Kajian ini memberi tumpuan kepada pengoptimuman ujian regresi dengan melaksanakan pemberat kes ujian dan kadar pengesanan kesalahan setiap kes ujian sebagai fungsi objektif untuk tujuan pengoptimuman. Teknik yang dicadangkan digunakan untuk kajian kes berasaskan GUI. Hasil kajian menunjukkan bahawa Pareto hadapan yang dihasilkan oleh NSGA II yang ditingkatkan memberikan satu set penyelesaian yang lebih luas yang mengandungi lebih banyak alternatif dan menyediakan penyelesaian yang lebih baik. Penilaian menunjukkan bahawa peningkatan NSGA II lebih baik berbanding dengan NSGA II konvensional dengan peningkatan peratusan kes ujian yang dikurangkan dengan 25% dan menghasilkan kehilangan pengesanan kesalahan yang lebih rendah dengan 1.64% yang menunjukkan bahawa kes ujian dikurangkan menggunakan NSGA II ditingkatkan dapat mengekalkan keupayaan pengesanan kesalahan dalam sistem yang diuji.

TABLE OF CONTENT

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

ABC	-	Artificial Bee Colony
ACO	-	Ant Colony Optimization
CS	-	Cuckoo Search
FAexGA	-	Fuzzy-Based Age Extension of Genetic Algorithm
GA	-	Genetic Algorithm
NSGA II		Non-dominated Sorting Genetic Algorithm
PSO	-	Particle Swarm Optimization
RWGA	-	Random Weight Genetic Algorithm
SDLC	-	Software Development Life Cycle
SSO	-	Simplified Swarm Optimization
SUT	-	System Under Test
WBGA	-	Weighted-Based Genetic Algorithm

LIST OF APPENDICES

CHAPTER 1

INTRODUCTION

1.1 Overview

Software testing is one of the important and crucial phase in software development life cycle. Generally, software testing purposes to ensure the correctness and error-free of the software that is developed. In the conventional Software Development Life Cycle (SDLC), software testing placed at fourth phase which means a particular software need to undergo all three phases before it can be tested by software testers. There are a lot of disadvantages for software testing in traditional SDLC. Throughout the years, software developers and software testers agreed that software testing needs to be done after each new iteration and changes occur in particular software. This situation leads to Agile development framework that overcomes major problems in software testing in traditional SDLC (Nidagundi and Novickis, 2017).

The basic concept of software testing can be understood by divided it into two groups which are black box testing and white box testing. Shortly, black box testing focusing on GUI of particular System Under Test (SUT) without considering internal structure of code for the software. Meanwhile, white box testing is the reverse concept which considering and testing the whole internal structure of codes of the software. Software testing can be executed manually or automatically. Some research claimed that automated testing is much better compared to manual testing (Sharma *et al.,*

2013). It is due to many problems in manual testing that can be reduced and completely overcome by using automated testing rather than manual testing. Another important testing process in software testing is regression testing. Regression testing can be exhaustive and expensive due to changes occurring in SUT (Zheng *et al.,* 2016). Regression testing is very important for current software development process which is iterative and continuous that need to be tested frequently. Manual and automated testing can be time-consuming in regression testing since the whole system needs to tested again even a small change occurred thus become the major drawback in regression testing. Hence, test case selection, optimization and prioritization have been introduced by multiple researchers in order to overcome the main problem in regression testing. Although automated and manual testing can be done parallelly, there are several aspects that need to be considered to ensure the effectiveness of the developed software. As mentioned previously, software testing is all about ensuring the correctness of software so that the software itself is working efficiently without error and be able to help to solve problems in many fields.

1.2 Problem Background

Software testing has its own life cycle which is also known as Software Testing Life Cycle (STLC). STLC indicates all the process involves in the testing process starting with requirement analysis, test planning, test case development, test environment setup, test execution and test cycle closure. The most critical part in STLC is test case development which indicates the activity of test suite, test cases and test data generation.

It is well known that software nowadays has improved in so many ways. Thus, the software itself has become more complex and complicated in order to develop, test and maintain the software. Rapid development and fast deployment of the software has become major concern of the customers hence giving a lot of pressure to the software development team to accomplish it. Some software that in the market today didn't perform software testing to their products that may lead to catastrophic effect

in the future. It is because software testing become time and cost consuming since the software that being developed has grown into complicated application.

Theoretically, larger and complex software system consist of many functions thus require more time and cost to undergo software testing process. The number of test cases generated also has increase in order to achieve full coverage of software testing for particular SUT. Hence, a smaller number of test cases which also a subset of original test cases that need to be executed has become major concern of software testers without neglecting full coverage criteria for particular SUT (Jeyaprakash and Alagarsamy, 2015). This situation is also known as test cases optimization, minimization and reduction. Test case optimization purposes to find the subset from set of test cases which contain the most optimized set of test cases by eliminating redundancy in test cases and selecting the best and have good criteria declared in particular test suite (Singh 2014). The problem concept of test case optimization can be derived as T = {T1, T2, T3,....Tn}, whereas T is the original test suite consists of larger number of test cases. Meanwhile, T' = {T1', T2', T3',....Tn'}, whereas T' consist of the most efficient test cases that optimized from original test suite (Chaudhary, 2016).

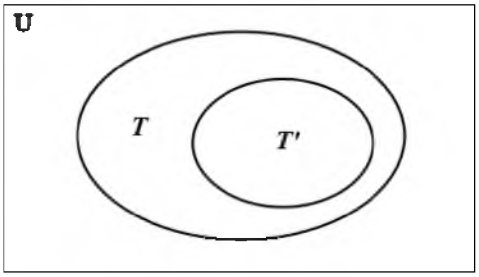

Figure 1.1 Concept of Test Case Optimization

In early introduction of test case optimization in regression testing, it is treated as single objective optimization problem which only considering reducing overall cost for software testing as their main objective. However, test case optimization is a technique that can provide trade-off between overall cost of software testing and other variables and factors to achieve more efficient software testing which lead to the introduction of multi-objective optimization problem (Savsani and Tawhid 2017). In

addition, multi-objective optimization problem is hard to satisfy all objectives since it has many optimal solutions for every objective (Cheng *et al.,* 2013).

Many researchers have treated test case optimization technique as a multi-objective optimization problem that considering more than two objectives in one time such as overall cost, fault detection capability or number of faults detected, reduced number of test cases and time taken for optimization technique. As a multi-objective problem, test case optimization needs to be flexible in finding as many as optimal solutions to solve the problem in software testing mainly in regression testing. Hence, Pareto optimal is introduced in order to find the optimal set of solutions that can be provided to solve test cases optimization problem (Chaudhary, 2016). By finding the Pareto set of solution, the objectives can be trade-off so that software testers can choose the best solution for the problems.

Many researches have been conducted in order to help software testers to fully optimize their test cases in test suite of software testing by implementing multiple types of algorithms and framework. Several popular algorithms such as Particle Swarm Optimization (PSO), Simplified Swarm Optimization (SSO), Artificial Bee Colony (ABC), Ant Colony Optimization (ACO), Cuckoo Search (CS) Algorithm and Genetic Algorithm (GA) has been implemented as optimization algorithm for test cases minimization and reduction.

All of these optimization algorithms are adapting the natural behaviour of the living organism in the world. For example, Artificial Bee Colony (ABC) is adapting the concept of bees in the real world. Three groups of bees are introduced in this approach which are scout bees, onlooker bees and employee bees (Lam *et al.,* 2012). Next, Cuckoo Search (CS) is adapting one type of bird named cuckoo which usually laying their eggs in other bird's nest. It will eliminate foreign eggs in the nest so that their eggs can survive for hatching (Ahmed *et al.,* 2015). On the other hands, Genetic Algorithm (GA) which is based on Darwin concept of evolution consist of population of chromosomes that needed for their next generation (Jeyaprakash and Alagarsamy, 2015). Next chapter of this research will explain briefly of all the algorithms and techniques mentioned previously.

Some of these techniques claimed to be effective and efficient in order to help in optimization of test cases. The main challenge of these algorithms is to find their fitness function to help to produce a set of optimized test case as an output. The main drawback in basic GA is that it is only implementing their own basic operation which includes selection, crossover and mutation of chromosomes which may lead to inefficient fitness function evaluation. However, Genetic Algorithm is found as relatively quite simple and effective based on previous researches. Hence, this research will focus on enhancing GA in particular approach to find better set of optimize test cases for particular SUT.

1.3 Problem Statement

The number of test cases in test suite will increase as the software that being developed evolve into much bigger and complicated software. Test cases optimization is one of the techniques to help increase the effectiveness of entire software testing itself. Basically, test cases optimization works by eliminating redundant test cases in a test suite and also finding the best set of test cases by considering the coverage of the test cases. Full coverage of software testing with less number of test cases has become a major concern in test case optimization technique.

In term of GUI testing, generated test cases will consist of events in particular software that may consist of redundant events and may increase overall execution cost. Some of the generated test cases may not be able to detect any fault in the software hence it is not required for GUI testing and can be eliminated from the test suite. However, eliminating such test cases may not an easy step. Software testers need to ensure that applied optimization technique did not exclude and ignores test cases that have longer number of events executed because it may reveal more faults in the SUT (Nguyen *et al.,* 2014).

Genetic Algorithm (GA) is one of the available techniques in order to find the most optimized set of test cases. The basic approach of GA in optimizing test cases starting with random generation of chromosomes that represent the population. Next,

the fitness value of each chromosome in the population is calculated in order to continue with selection, crossover and mutation operators to generate new population which more fit and optimized (Singhal *et al.,* 2012). Stopping criteria is applied to the population to determine whether the new population is achieved the targeted fitness value.

Multiple variants of GA have been implemented throughout the years as test cases optimization algorithm such as Weight-Based Genetic Algorithm (WBGA), Fuzzy-Based Age Extension of Genetic Algorithm (FAexGA) and Non-Dominated Sorting Genetic Algorithm (NSGA II). In general, WBGA implements weight on the chromosomes to find the fitness value hence lead to optimizing overall test cases(Wang *et al.,* 2013). Meanwhile, FAexGA purposes to assign aging technique to the test cases to eliminate the old test cases (Last *et al.,* 2006). Each of these algorithms has their own drawback. For example, WBGA cannot become the best solution for test case optimization due to fixed weight applied to the test cases while FAexGA techniques only applicable to GUI testing only. NSGA II on the other hand is sorting the test cases using only crowding distance approach to find the most optimize set of test cases (Jeyaprakash and Alagarsamy, 2015).

Hence, this research focuses on the implementation of Genetic Algorithm (GA) as the main algorithm for test case optimization. Non-Dominated Sorting Genetic Algorithm (NSGA II) is chosen as the main algorithm and also the implementation of the basic concept of GA itself which is crossover, mutation and fitness scaling as evaluation of fitness function. Most of the research that implementing NSGA II only depending on Pareto-Ranking function and additional fitness function to obtain most optimized set of test cases in the particular test suite. This research however tries to extend NSGA II by implementing fitness scaling process that may produce more efficient set of test cases. According to the statement provided previously, main research question can be derived as follow:

"How to increase the percentage of reduced number of test cases in test case optimization using Non-Dominated Sorting Genetic Algorithm (NSGA II)?"

Based on main research question above, several minor research questions can be constructed in order to answer the main question. Minor research questions constructed as follow:

i. What is a better technique for determining fitness value of test cases apart from Pareto ranking approach in NSGA II?
ii. How may the identified fitness scaling technique improve the percentage of reduce number of test cases?
iii. How to evaluate the effectiveness of the identified technique in reducing the number of test cases?

1.4 Research Aim and Objectives

This research aims to enhance existing optimization technique in regression GUI testing using Non-Dominated Sorting Genetic Algorithm (NSGA II) by implementing fitness scaling approach alongside with Pareto-Ranking approach to the algorithm which may produce more optimize GUI test cases. Based on the research aim mentioned, several objectives are generated as guidance for this research. The objectives of this research are:

i. To improve the Non-Dominated Sorting Genetic Algorithm (NSGA II) for test cases optimization technique.
ii. To evaluate the improved technique by benchmarking with the original algorithm of NSGA II.

1.5 Scope of Study

This study is targeting in the optimization of test cases for small and medium size of software that requires rapid software testing process which consists of large number of test cases in their test suite. The paramount of this study are as follow:

i. This study focuses on the implementation of fitness scaling techniques in Non-Dominated Sorting Genetic Algorithm (NSGA II) in order to find the most optimize test cases in the test suite.
ii. Main priorities of the output expected from this study are increasing the percentage of the reduced number of test cases and maintaining fault detection capability of test case in particular software testing.
iii. It is also focusing on enhancing the existing algorithm and comparing it with the original one.

1.6 Significance of Study

This research purposes to give benefits to the software testing industry focusing on GUI regression testing by providing another alternative solution for test cases optimization alongside with other optimization techniques available in the field. In addition, this approach proposed in this study may help small and medium size of software application for fast software testing before a particular software being released to the market. Moreover, this approach also may contribute an additional knowledge of software testing on the body of knowledge for students and other researchers to contribute more in test case optimization technique.

1.7 Dissertation Organization

This thesis consists of five chapter. The first chapter states an overview of software testing, Software Testing Life Cycle (STLC) and the overall description of propose technique implemented in this research. Chapter two explains the details of software testing, STLC and previous techniques implemented as test case optimization algorithms. Next, the concept of implementation of Non-Dominated Sorting Genetic Algorithm (NSGA II) and the framework of execution is described in Chapter three. Chapter four of this thesis include early result and discussion of proposed technique. The final chapter of this thesis will conclude the proposed technique in term of effectiveness and efficiency of test case optimization using enhancement of NSGA II.

REFERENCES

AdiSrikanth, Kulkarni, N., Naveen, K., Singh, P. and Srivastava, P. (2011). Test Case Optimization Using Artificial Bee Colony Algorithm. *Advances in Computing and Communications*, 333031, 570–579.

Ahmed, B.S. (2016). Test case minimization approach using fault detection and combinatorial optimization techniques for configuration-aware structural testing. *Engineering Science and Technology, an International Journal*, 19(2), 737–753.

Ahmed, B.S., Abdulsamad, T.S. and Potrus, M.Y. (2015). Achievement of minimized combinatorial test suite for configuration-aware software functional testing using the Cuckoo Search algorithm. *Information and Software Technology*, 66, 13–29.

Ahmed, B.S., Sahib, M.A. and Potrus, M.Y. (2014). Generating combinatorial test cases using Simplified Swarm Optimization (SSO) algorithm for automated GUI functional testing. *Engineering Science and Technology, an International Journal*, 17(4), 218–226.

Belli, F., Beyazit, M. and Güler, N. (2011). Event-Based GUI Testing and Reliability Assessment Techniques–An Experimental Insight and Preliminary Results. *Software Testing, Verification and Validation Workshops (ICSTW), 2011 IEEE Fourth International Conference on*, 212–221.

Chaudhary, N. (2016). Multi Objective Test Suite Reduction for GUI Based Software Using NSGA-II. , (August), 59–65.

Chaudhary, N., Sangwan, O.P. and Arora, R. (2014). Event-coverage and weight based method for test suite prioritization. *International Journal of Information Technology and Computer Science (IJITCS)*, 6(12), 61.

Cheng, L., Tsou, C., Lee, M., Huang, L., Song, D. and Teng, W. (2013). Tradeoff analysis for optimal multiobjective inventory model. *Journal of Applied Mathematics*, 2013(i).

Day, P. (2014). N-Tiered test automation architecture for Agile software systems. *Procedia Computer Science*, 28, 332–339.

Deb, K., Pratab, S., Agarwal, S. and Megarivan, T. (2002). A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6(2), 182–197.

Elbaum, S., Rothermel, G. and Penix, J. (2014). Techniques for improving regression testing in continuous integration development environments. *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering - FSE 2014*, 235–245.

Holst, T.L. (2005). Genetic Algorithms Applied to Multi-Objective Aerospace Shape Optimization. *Journal of Aerospace Computing, Information, and Communication*, 2(4), 217–235.

Huang, C.Y., Chen, C.S. and Lai, C.E. (2016). Evaluation and analysis of incorporating Fuzzy Expert System approach into test suite reduction. *Information and Software Technology*, 79, 79–105.

Jeyaprakash, S. and Alagarsamy, K. (2015). A distinctive genetic approach for test-suite optimization. *Procedia Computer Science*, 62(Scse), 427–434.

Kandil, P., Moussa, S. and Badr, N. (2014). Regression Testing Approach for Large-Scale Systems. *2014 IEEE International Symposium on Software Reliability Engineering Workshops*, 132–133.

Kumar, G. and Bhatia, P.K. (2016). Software Test Case Reduction using Genetic Algorithm : A Modified Approach. , 3(5), 349–354.

Kumar, G. and Bhatia, P.K. (2013). Software testing optimization through test suite reduction using fuzzy clustering. *CSI Transactions on ICT*, 1(3), 253–260.

Kumari, A.C. (2013). RegressAid – A CASE Tool for Minimization of Test Suite for Regression Testing. , 71(18), 30–34.

Lam, S.S.B., Raju, M.LH.P., M, U.K., Ch, S., Srivastav, P.R., and Sekhara, S. (2012). Automated Generation of Independent Paths and Test Suite Optimization Using Artificial Bee Colony. *Procedia Engineering*, 30(2011), 191–200.

Last, M., Eyal, S. and Kandel, A. (2006). Effective Black-Box Testing with Genetic Algorithms. , 134–148.

Marchetto, A. and Islam, M. (2013). A Multi-Objective Technique for Test Suite Reduction. *ICSEA 2013, The Eighth ...*, (c), 18–24.

Mateen, A. (2016). Optimization of Test Case Generation using Genetic Algorithm (GA). , 151(7), 6–14.

McMaster, S. and Memon, A. (2008). Call-stack coverage for GUI test suite reduction. *IEEE Transactions on Software Engineering*, 34(1), 99–115.

Memon, A.M., Soffa, M. Lou and Pollack, M.E. (2001). Coverage criteria for GUI testing. *ACM SIGSOFT Software Engineering Notes*, 26(5), 256.

Memon, a. M., Pollack, M.E. and Soffa, M.L. (2001). Hierarchical GUI test case generation using automated planning. *IEEE Transactions on Software Engineering*, 27(2), 144–155.

Mondal, D., Hemmati, H. and Durocher, S. (2015). Exploring test suite diversification and code coverage in multi-objective test case selection. *2015 IEEE 8th International Conference on Software Testing, Verification and Validation, ICST 2015 - Proceedings*.

Nguyen, B.N., Robbins, B., Banarjee, I. and Memon, A. (2014). GUITAR: An innovative tool for automated testing of GUI-driven software. *Automated Software Engineering*, 21(1), 65–105.

Nidagundi, P. and Novickis, L. (2017). Introducing Lean Canvas Model Adaptation in the Scrum Software Testing. *Procedia Computer Science*, 104(December 2016), 97–103.

Nidagundi, P. and Novickis, L. (2016). Introduction to Lean Canvas Transformation Models and Metrics in Software Testing. *Applied Computer Systems*, 19(1), 30–36.

de Oliveira Neto, F.G., Torkar, R. and Machado, P.D.L. (2016). Full modification coverage through automatic similarity-based test case selection. *Information and Software Technology*, 80, 124–137.

Samatha, K., Chokkadi, S. and Yogananda, J. (2012). A Genetic Algorithm Approach for Test Case Optimization of Safety Critical Control. *Procedia Engineering*, 38, 647–654.

Sanchez, A.B., Segura, S. and Ruiz-Cortes, A. (2014). A Comparison of Test Case Prioritization Criteria for Software Product Lines. *2014 IEEE Seventh International Conference on Software Testing, Verification and Validation*, 41–50.

Savsani, V. and Tawhid, M.A. (2017). Non-dominated sorting moth flame optimization (NS-MFO) for multi-objective problems. *Engineering Applications of Artificial Intelligence*, 63, 20–32.

Schwartz, A. and Do, H. (2016). Cost-effective regression testing through Adaptive Test Prioritization strategies. *Journal of Systems and Software*, 115, 61–81.

Sharma, C., Sabharwal, S. and Sibal, R. (2013). A Survey on Software Testing Techniques using Genetic Algorithm. *International Journal of Computer Science Issues*, 10(1), 381–393.

Singh, R. (2014). Test Suite Minimization using Evolutionary Optimization Algorithms: Review. , 3(6), 2086–2091.

Singhal, A., Chandna, S. and Bansal, A. (2012). Optimization of Test Cases Using Genetic Algorithm 1. , 2(3), 367–369.

De Souza, L.S., Prudencio, R.B.C., Barros, F.D.A. and Aranha, E.H.D. (2013). Search based constrained test case selection using execution effort. *Expert Systems with Applications*, 40(12), 4887–4896.

Srividhya, J. (2014). A Synthesized Overview of Test Case Optimization Techniques. , 1(2).

Wang, S., Ali, S. and Gotlieb, A. (2013). Minimizing test suites in software product lines using weight-based genetic algorithms. *2013 15th Genetic and Evolutionary Computation Conference, GECCO 2013*, 1493–1500.

Yoo, S. and Harman, M. (2007). Pareto efficient multi-objective test case selection. *Proceedings of the 2007 international symposium on Software testing and analysis - ISSTA '07*, 140.

Yoo, S. and Harman, M. (2014). Regression testing minimization, selection and prioritization: A Survey. *Software Testing Verification and Reliability*, 24(8), 591–592.

Yusoff, Y., Ngadiman, M.S. and Zain, A.M. (2011). Overview of NSGA-II for optimizing machining process parameters. *Procedia Engineering*, 15, 3978–3983.

Zheng, W., Hierons, R.M., Li, M., Liu, X. and Vinciotti, V. (2016). Multi-objective optimisation for regression testing. *Information Sciences*, 334, 1–16.