REPETITIVE MUTATIONS IN GENETIC ALGORITHM FOR SOFTWARE TEST DATA GENERATIONS

MOHAMMED MAJID KADHIM

MASTER OF INFORMATION TECHNOLOGY
UNIVERSITI UTARA MALAYSIA
2022

KOLEJ SASTERA DAN SAINS
(College of Arts and Sciences)
Universiti Utara Malaysia

PERAKUAN DISERTASI
(Certificate of Dissertation)

Saya, yang bertandatangan, memperakukan bahawa
(I, the undersigned, certifies that)

MOHAMMED MAJID KADHIM
(827647)

calon untuk Ijazah
(candidate for the degree of) **MSc. (Information Technology)**

telah mengemukakan disertasi yang bertajuk
(has presented his/her dissertation of the following title)

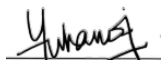
Repetitive Mutations in Genetic Algorithm for Software Test Data Generations

seperti yang tercatat di muka surat tajuk dan kulit disertasi
(as it appears on the title page and front cover of dissertation)

bahawa disertasi tersebut boleh diterima dari segi bentuk serta kandungan dan meliputi bidang ilmu dengan memuaskan.
(that this dissertation is in acceptable form and content, and that a satisfactory knowledge of the field is covered by the dissertation).

Nama Penyelia (1) Tandatangan:
Name of Supervisor (1) : DR ZHAMRI CHE ANI *(Signature)* _____

Nama Penyelia (2) Tandatangan:
Name of Supervisor (2) : PROF MADYA DR MAZNI OMAR *(Signature)* _____

Tandatangan: 

Nama Penilai (1:)
Name of Evaluator (1): PROF MADYA DR YUHANIS YUSOF (Signature)

Nama Penilai (2) Tandatangan: 
Name of Evaluator (2): PROF MADYA DR AZMAN YASIN (*Signature)* _____

Tarikh (Date): 23/8/2022

Permission to Use

In presenting this thesis in fulfilment of the requirements for a postgraduate degree from Universiti Utara Malaysia, I agree that the Universiti Library may make it freely available for inspection. I further agree that permission for the copying of this thesis in any manner, in whole or in part, for scholarly purpose may be granted by my supervisor(s) or, in their absence, by the Dean of Awang Had Salleh Graduate School of Arts and Sciences. It is understood that any copying or publication or use of this thesis or parts thereof for financial gain shall not be allowed without my written permission. It is also understood that due recognition shall be given to me and to Universiti Utara Malaysia for any scholarly use which may be made of any material from my thesis.

Requests for permission to copy or to make other use of materials in this thesis, in whole or in part, should be addressed to:

Dean of Awang Had Salleh Graduate School of Arts and Sciences
UUM College of Arts and Sciences
Universiti Utara Malaysia
06010 UUM Sintok

Abstrak

Penjanaan data ujian adalah penting dalam pengujian perisian dinamik. Salah satu teknik pengujian kotak putih ialah pengujian liputan laluan. Algoritma Genetik (GA) telah terbukti sebagai kaedah penting dalam menjana data ujian untuk pengujian liputan laluan automatik. Walau bagaimanapun, untuk memenuhi ujian liputan laluan, operasi mutasi tunggal GA menjana data ujian yang meliputi laluan yang sama dalam satu generasi, justeru mengakibatkan pertindihan liputan laluan, yang secara negatif meningkatkan bilangan lelaran. Oleh itu, kajian ini mencadangkan teknik mutasi berulang GA untuk menghapuskan pertindihan liputan laluan dan mengurangkan bilangan lelaran bagi penjanaan data ujian dalam pengujian liputan laluan. Kajian ini dilaksanakan dalam tiga fasa. Pertama, limitasi teknik mutasi sedia ada yang digunakan dalam GA untuk menjana data ujian untuk pengujian liputan laluan dianalisa. Kemudian, teknik mutasi berulang untuk GA direka bentuk dan dilaksanakan dalam simulasi berangka menggunakan bahasa C++. Akhir sekali, fasa penilaian yang membandingkan hasil teknik yang dicadangkan dengan kajian sedia ada dari segi bilangan lelaran untuk penjanaan data ujian. Hasil kajian menunjukkan bahawa pendekatan mutasi berulang yang dicadangkan mengatasi teknik mutasi tunggal dengan pengurangan bilangan lelaran melebihi 50 peratus untuk penjanaan data ujian. Kajian ini menunjukkan kepentingan mutasi dalam penjanaan data ujian dan bagaimana ia dapat dimanfaatkan untuk membolehkan GA mendapatkan penyelesaian dengan lebih cepat. Selain itu, cadangan mutasi berulang GA boleh menyumbang kepada pembangunan alatan ujian GA adaptif.

Kata kunci: Pengujian liputan laluan, **A**lgoritma genetik, Mutasi berulang, Ujian penjanaan data.

Abstract

Generating test data is the most important part of dynamic software testing. One of the white box testing techniques is path coverage testing. Genetic Algorithm (GA) has proven to be an important method in generating test data for automatic path coverage testing. However, to satisfy path coverage testing, GA's operation of a single mutation generates test data that covers the same path in a single generation, hence resulting in path coverage duplication, which negatively increases the number of iterations. Therefore, this study proposes a repetitive mutation for GA in order to eliminate path coverage duplication and reduce the number of iterations for test data generations in path coverage testing. The study was conducted in three phases. First, the limitations of existing mutation techniques used in GA to generate test data for path coverage testing were analysed. Then, a repetitive mutation technique for GA was designed and implemented in a numerical simulation using C++ language. Finally, the evaluation phase that compares the outcome of the proposed technique against existing studies in terms of the number of iterations for test data generations. The findings show that the proposed repetitive mutation technique outperformed the single mutation technique by reducing the number of iterations to more than 50 percent for test data generations. The study has revealed the importance of mutation in generating test data and how it can be harnessed to quickly guide GA in producing solutions. In addition, the proposed repetitive mutation in GA can contribute to developing an adaptive GA testing tool.

Keywords: Path coverage testing, Genetic algorithm, Repetitive mutations, Test data generations.

Acknowledgement

Thank Allah the great for his care, kindness, and support in completing my thesis. I want to thank my Supervisors, Dr. Zhamri Bin Che Ani and Assoc Prof. Dr. Mazni Binti Omar, for their continued support, advice, comments, kindness, and encouragement during my study journey. They have devoted their knowledge and time to guiding me toward this level. My study would not have been achieved without them. Thank you very much, and all your efforts are appreciated forever.

I would also like to thank my parents, wife, and family for their moral support and patience.

I am very grateful to Assoc Prof. Dr. Rohaida Romli, Assoc Prof. Dr. Azman Yasin, and Assoc Prof. Dr. Yuhanis Yusof for their comments and advices, which have helped improve this work a lot.

I also cannot forget the helpful coordination and assistance provided by Dr. Juliana Binti Wahid and Dr. Farzana Binti Kabir Ahmad.

It was an amazing experience to study MSc at Universiti Utara Malaysia (UUM).

Table of Contents

List of Tables

List of Figures

List of Appendices

List of Abbreviations

ACO	Ant Colony Optimization
AI	Artificial Intelligence
CDC	Check Duplication Count
CFG	Control Flow Graph
ES	Evolutionary Strategy
ESALOGA	Enhanced Selection and Log-scaled Mutation GA
GA	Genetic Algorithm
GABVIE	Genetic Algorithm Based Variable Importance Evaluation
GCD	Greatest Common Divisor
GSA	Gravitational Search Algorithm
IDE	Integrated Development Environment
MCDC	Modified Condition Decision Coverage
NSA	Negative Selection Algorithm
PSO	Particle Swarm Optimization
SA	Simulated Annealing
SBM	Select the Best Mutation Algorithm
SDLC	Software Development Life Cycle
SUT	Software Under Test
TS	Tabu Search
TSP	Traveling Salesman Problem
UAT	User Acceptance Testing
WOA	Whale Optimization Algorithm

CHAPTER ONE

INTRODUCTION

1.1 Overview

The background of the study is introduced in this chapter, followed by a discussion of the research problem. Then, the research questions are presented and used to formulate the research objectives. Finally, the chapter highlights the significance of the study, describes the scope, illustrates the operational definitions, and describes the organisation of the thesis.

1.2 Background of the Study

Automatic systems are ubiquitous, in which they can be controlled and managed using their software. For example, experts have developed software for mobile phones, cars, houses, hospitals, etc.; however, maintaining its quality is not an easy task (Khan & Amjad, 2016b). The most significant phase in any Software Development Life Cycle (SDLC) is software testing. Qualified and reliable software must undergo good testing (Mustafa et al., 2021). Occasionally, poorly tested software may function well for months and years, even if it has bugs, but input sets may discover serious errors at any moment. Software released in the market without good testing causes customer dissatisfaction and a bad company reputation, leading to a series of financial losses or even endangering the lives of human beings (Majma & Babamir, 2014). Hence, the main goal of software testing is to provide confidence in the correctness of the system.

REFERENCES

Abdoun, O., Abouchabaka, J., & Tajani, C. (2012). Analyzing the performance of mutation operators to solve the travelling salesman problem. *International Journal of Emerging Sciences*, *2*(1), 61–77. http://arxiv.org/abs/1203.3099

Agnihotri, J., & Kumar, V. (2015). A study of various metaheuristic techniques used for software testing. *International Journal of Advanced Trends in Computer Applications*, *1*(5), 39–45. https://www.academia.edu/34984165/A_Study_of_Various_Metaheuristic_Techniques_used_for_Software_Testing

Ahmed, M. A., & Hermadi, I. (2008). GA-based multiple paths test data generator. *Computers and Operations Research*, *35*(10), 3107–3124. https://doi.org/10.1016/j.cor.2007.01.012

Anh, B. T. M. (2020). Enhanced genetic algorithm for automatic generation of unit and integration test suite. In *Proceedings - 2020 RIVF International Conference on Computing and Communication Technologies, RIVF 2020* (pp. 1–6). https://doi.org/10.1109/RIVF48685.2020.9140778

Anwar, N., & Kar, S. (2019). Review paper on various software testing techniques & strategies. *Global Journal of Computer Science and Technology*, *19*(2), 43–49. https://doi.org/10.34257/gjcstcvol19is2pg43

Avdeenko, T., & Serdyukov, K. (2021). Automated test data generation based on a genetic algorithm with maximum code coverage and population diversity. *Applied Sciences (Switzerland)*, *11*(10). https://doi.org/10.3390/app11104673

Aysolmaz, B., Leopold, H., Reijers, H. A., & Demirörs, O. (2018). A semi-automated approach for generating natural language requirements documents based on business process models. *Information and Software Technology*, *93*, 14–29. https://doi.org/10.1016/j.infsof.2017.08.009

Bahaweres, R. B., Zawawi, K., Khairani, D., & Hakiem, N. (2017). Analysis of statement branch and loop coverage in software testing with genetic algorithm. *2017 4th International Conference on Electrical Engineering, Computer Science and Informatics (EECSI)*, *September*, 1–6. https://doi.org/10.1109/EECSI.2017.8239088

Bandaru, S., & Deb, K. (2016). Metaheuristic techniques. In R. N. Sengupta, A. Gupta, & J. Dutta (Eds.), *Decision Sciences: Theory and Practice* (pp. 693–750). CRC Press. https://doi.org/10.1201/9781315183176

Bansal, J. C. (2019). Particle swarm optimization. In J. Bansal, P. Singh, & N. Pal (Eds.), *Evolutionary and swarm intelligence algorithms – Studies in computational intelligence* (pp. 11–23). Springer. https://doi.org/https://doi.org/10.1007/978-3-319-91341-4_2

Bashir, M. B., & Nadeem, A. (2017). Improved genetic algorithm to reduce mutation testing cost. *IEEE Access*, *5*, 3657–3674. https://doi.org/10.1109/ACCESS.2017.2678200

Beg, A. H., & Islam, M. Z. (2016). Novel crossover and mutation operation in genetic algorithm for clustering. *2016 IEEE Congress on Evolutionary Computation, CEC 2016*, *July*, 2114–2121. https://doi.org/10.1109/CEC.2016.7744049

Berndt, D. J., & Watkins, A. (2005). High volume software testing using genetic

algorithms. *Proceedings of the 38th Annual Hawaii International Conference on System Sciences*, 318a–318b. https://doi.org/10.1109/hicss.2005.296

Boopathi, M., Sujatha, R., Kumar, C. S., Narasimman, S., & Rajan, A. (2019). Markov approach for quantifying the software code coverage using genetic algorithm in software testing. *International Journal of Bio-Inspired Computation*, *14*(1), 27–45. https://doi.org/10.1504/IJBIC.2019.101152

Chen, C., Xu, X., Chen, Y., Li, X., & Guo, D. (2009a). A new method of test data generation for branch coverage in software testing based on EPDG and genetic algorithm. *2009 3rd International Conference on Anti-Counterfeiting, Security, and Identification in Communication (ASID)*, 307–310. https://doi.org/10.1109/ICASID.2009.5276897

Chen, Y., Zhong, Y., Shi, T., & Liu, J. (2009b). Comparison of two fitness functions for GA-based path-oriented test data generation. *2009 5th International Conference on Natural Computation (ICNC)*, *4*, 177–181. https://doi.org/10.1109/ICNC.2009.235

Dasoriya, R., & Dashoriya, R. (2018). Use of optimized genetic algorithm for software testing. *2018 IEEE International Students' Conference on Electrical, Electronics and Computer Science, SCEECS 2018*, 1–5. https://doi.org/10.1109/SCEECS.2018.8546957

de Freitas, F. G., Maia, C. L. B., de Campos, G. A. L., & de Souza, J. T. (2010). Optimization in software testing using metaheuristics. *Revista de Sistemas de Informação Da FSMA*, *5*, 3–13. http://www.fsma.edu.br/si/edicao5/FSMA_SI_2010_1_Estudantil_1_en.pdf

de Jong, K. A. (1975). *Analysis of the behavior of a class of genetic adaptive systems*. [Doctoral dissertation, University of Michigan].

Deb, K. (2012). *Optimization for engineering design: Algorithms and examples.* (2nd ed.). PHI Learning.

Deep, K., & Thakur, M. (2007). A new crossover operator for real coded genetic algorithms. *Applied Mathematics and Computation*, *188*(1), 895–911. https://doi.org/10.1016/j.amc.2006.10.047

Delahaye, D., Chaimatanan, S., & Mongeau, M. (2019). Simulated annealing: From basics to applications. In M. Gendreau & J.-Y. Potvin (Eds.), *Handbook of metaheuristics* (pp. 1–35). Springer. https://hal-enac.archives-ouvertes.fr/hal-01887543/document

Duan, P., & Yong, A. I. (2016). Research on an improved Ant Colony Optimization Algorithm and its application. *International Journal of Hybrid Information Technology*, *9*(4), 223–234. https://gvpress.com/journals/IJHIT/vol9_no4/20.pdf

Felderer, M., & Ramler, R. (2014). Integrating risk-based testing in industrial test processes. *Software Quality Journal*, *22*(3), 543–575. https://doi.org/10.1007/s11219-013-9226-y

Garg, D., & Garg, P. (2015). Basis path testing using SGA & HGA with ExLB fitness function. *Procedia Computer Science*, *70*, 593–602. https://doi.org/10.1016/j.procs.2015.10.044

Ghani, K., & Clark, J. A. (2009). Automatic test data generation for multiple condition and MCDC coverage. *2009 Fourth International Conference on Software Engineering Advances*, 152–157. https://doi.org/10.1109/ICSEA.2009.31

Gharehchopogh, F. S., & Gholizadeh, H. (2019). A comprehensive survey: Whale optimization algorithm and its applications. *Swarm and Evolutionary Computation*, *48*, 1–24. https://doi.org/https://doi.org/10.1016/J.SWEVO.2019.03.004

Ghiduk, A. S. (2014). Automatic generation of basis test paths using variable length genetic algorithm. *Information Processing Letters*, *114*(6), 304–316. https://doi.org/10.1016/j.ipl.2014.01.009

Girgis, M. R., Ghiduk, A. S., & Abd-Elkawy, E. H. (2014). Automatic generation of data flow test paths using a genetic algorithm. *International Journal of Computer Applications*, *89*(12), 29–36. https://research.ijcaonline.org/volume89/number12/pxc3894534.pdf

Godboley, S., Mohapatra, D. P., Das, A., & Mall, R. (2017). An improved distributed concolic testing approach. *Software - Practice and Experience*, *47*(2), 311–342. https://doi.org/https://doi.org/10.1002/spe.2405

Gong, D., Tian, T., & Yao, X. (2012). Grouping target paths for evolutionary generation of test data in parallel. *Journal of Systems and Software*, *85*(11), 2531–2540. https://doi.org/https://doi.org/10.1016/j.jss.2012.05.071

Gupta, K. D., & Dasgupta, D. (2021). Negative selection algorithm research and applications in the last decade: A review. *IEEE Transactions on Artificial Intelligence*, *3*(2), 110–128. https://doi.org/10.1109/TAI.2021.3114661

Gupta, N., Patel, N., Tiwari, B. N., & Khosravy, M. (2019). Genetic algorithm based on enhanced selection and log-scaled mutation technique. In K. Arai, R. Bhatia, & S. Kapoor (Eds.), *Proceedings of the Future Technologies Conference (FTC) 2018*

– *Advances in Intelligent Systems and Computing* (Vol. 880, pp. 333–342). Springer. https://doi.org/https://doi.org/10.1007/978-3-030-02686-8_55

Hassanat, A. B. A., Alkafaween, E., Al-Nawaiseh, N. A., Abbadi, M. A., Alkasassbeh, M., & Alhasanat, M. B. (2016). Enhancing genetic algorithms using multi mutations : Experimental results on the travelling salesman problem. *International Journal of Computer Science and Information Security*, *14*(7), 785–801. https://arxiv.org/ftp/arxiv/papers/1602/1602.08313.pdf

Hegazy, A. E., Makhlouf, M. A., & El-Tawel, G. (2018). Dimensionality reduction using an improved whale optimization algorithm for data classification. *International Journal of Modern Education and Computer Science*, *10*(7), 37–49. https://doi.org/10.5815/ijmecs.2018.07.04

Hermadi, I., Lokan, C., & Sarker, R. (2010). Genetic algorithm based path testing: Challenges and key parameters. *2010 Second World Congress on Software Engineering*, *2*, 241–244. https://doi.org/10.1109/WCSE.2010.82

Hermadi, I., Lokan, C., & Sarker, R. (2014). Dynamic stopping criteria for search-based test data generation for path testing. *Information and Software Technology*, *56*(4), 395–407. https://doi.org/10.1016/j.infsof.2014.01.001

Jaffari, A., Yoo, C. J., & Lee, J. (2020). Automatic test data generation using the activity diagram and search-based technique. *Applied Sciences (Switzerland)*, *10*(10), 9–13. https://doi.org/10.3390/APP10103397

Jena, T., & Mohanty, J. R. (2016). Disaster recovery services in intercloud using genetic algorithm load balancer. *International Journal of Electrical and Computer Engineering*, *6*(4), 1828–1838. https://doi.org/10.11591/ijece.v6i4.9956

Jindal, T. (2016). Importance of testing in SDLC. *International Journal of Engineering and Applied Computer Science*, *01*(02), 54–56. https://doi.org/10.24032/ijeacs/0102/05

Jovanovic, I. (2009). Software testing methods and techniques. In *The IPSI BgD transactions on internet research* (Vol. 5, pp. 30–41). http://vipsi.org/ipsi/journals/journals/tir/2009/January/Full Journal.pdf#page=31

Katoch, S., Chauhan, S. S., & Kumar, V. (2021). A review on genetic Aalgorithm: Past, present, and future. *Multimedia Tools and Applications*, *80*(5), 8091–8126. https://doi.org/10.1007/s11042-020-10139-6

Khan, M. E. (2011). Different approaches to black box testing technique for finding errors. *International Journal of Software Engineering and Its Applications*, *2*(4), 31–40. https://doi.org/10.5121/ijsea.2011.2404

Khan, M. E., & Khan, F. (2012). A comparative study of white box , black box and grey box testing techniques. *International Journal of Advanced Computer Science and Applications*, *3*(6), 12–15. https://doi.org/http://dx.doi.org/10.14569/IJACSA.2012.030603

Khan, R., & Amjad, M. (2016a). Automatic generation of test cases for data flow test paths using K-means clustering and generic algorithm. *International Journal of Applied Engineering Research*, *11*(1), 473–478. https://www.researchgate.net/profile/Rizwan-Khan-27/publication/298711135_Automatic_generation_of_test_cases_for_data_flow_test_paths_using_K means_clustering_and_generic_algorithm/links/581734dc08ae90acb24265c8/Au

tomatic-generation-of-test-cases-for-data

Khan, R., & Amjad, M. (2016b). Optimize the software testing efficiency using genetic algorithm and mutation analysis. *2016 3rd International Conference on Computing for Sustainable Global Development (INDIACom)*, *April*, 1174–1176.

Khan, R., & Amjad, M. (2016c). Performance testing (load) of web applications based on test case management. *Perspectives in Science*, *8*, 355–357. https://doi.org/10.1016/j.pisc.2016.04.073

Khan, R., & Amjad, M. (2017). Introduction to data flow testing with genetic algorithm. *International Journal of Computer Applications*, *170*(5), 39–45. https://doi.org/10.5120/ijca2017914845

Khan, R., Amjad, M., & Srivastava, A. K. (2016). Optimization of automatic generated test cases for path testing using genetic algorithm. *2016 Second International Conference on Computational Intelligence & Communication Technology (CICT)*, 32–36. https://doi.org/10.1109/CICT.2016.16

Khan, R., & Srivastava, A. K. (2019). Automatic software testing framework for all Def-Use with genetic algorithm. *International Journal of Innovative Technology and Exploring Engineering*, *8*(8), 2055–2060.

Li, X., & Yang, G. (2016). Artificial bee colony algorithm with memory. *Applied Soft Computing*, *41*(C), 362–372. https://doi.org/https://doi.org/10.1016/j.asoc.2015.12.046

Majma, N., & Babamir, S. M. (2014). Medical software runtime checking using Petri-nets & software agents. *2014 4th International Conference on Computer and Knowledge Engineering (ICCKE)*, *December*, 449–454.

https://doi.org/10.1109/ICCKE.2014.6993410

Malaguti, E., Martello, S., & Santini, A. (2018). The traveling salesman problem with pickups, deliveries, and draft limits. *Omega*, *74*(C), 50–58. https://doi.org/10.1016/j.omega.2017.01.005

Manikumar, T., & Kumar, A. (2019). A buffered genetic algorithm for automated branch coverage in software testing. *Journal of Information Science and Engineering*, *35*(2), 245–273. https://doi.org/10.6688/JISE.201903_35(2).0001

Mathur, A. P. (2013). *Foundations of software testing 2E*. https://www.cs.purdue.edu/homes/apm/FoundationsBookSecondEdition/Slides/ConsolidatedSlides.pdf

Minj, J., & Belchanden, L. (2013). Path oriented test case generation for UML state diagram using genetic algorithm. *International Journal of Computer Applications*, *82*(7), 1–4. https://doi.org/10.5120/14125-9813

Mirjalili, S., Faris, H., & Aljarah, I. (2020). Introduction to evolutionary machine learning techniques. In S. Mirjalili, H. Faris, & I. Aljarah (Eds.), *Evolutionary machine learning techniques – Algorithms and application* (pp. 1–7). Springer. https://doi.org/https://doi.org/10.1007/978-981-32-9990-0

Mishra, D. B., Acharya, A. A., & Acharya, S. (2020). White box testing using genetic algorithm – An extensive study. In J. Singh, S. Bilgaiyan, B. Mishra, & S. Dehuri (Eds.), *A journey towards bio-inspired techniques in software engineering – Intelligent systems reference library* (pp. 167–185). Springer. https://doi.org/https://doi.org/10.1007/978-3-030-40928-9_9

Mishra, D. B., Mishra, R., Acharya, A. A., & Das, K. N. (2019a). Test data generation

for mutation testing using genetic algorithm. In J. Bansal, K. Das, A. Nagar, K. Deep, & A. Ojha (Eds.), *Soft computing for problem solving – Advances in intelligent systems and computing* (Vol. 817, pp. 857–867). Springer. https://doi.org/10.1007/978-981-13-1595-4_68

Mishra, D. B., Mishra, R., Bilgaiyan, S., Acharya, A. A., & Mishra, S. (2017a). A review of random test case generation using genetic algorithm. *Indian Journal of Science and Technology*, *10*(30), 1–7. https://doi.org/https://dx.doi.org/10.17485/ijst/2017/v10i30/107654

Mishra, D. B., Mishra, R., Das, K. N., & Acharya, A. A. (2017b). A systematic review of software testing using evolutionary techniques. *Proceedings of Sixth International Conference on Soft Computing for Problem Solving – Advances in Intelligent Systems and Computing*, *546*, 174–184. https://doi.org/10.1007/978-981-10-3322-3_16

Mishra, D. B., Mishra, R., Das, K. N., & Acharya, A. A. (2019a). Test case generation and optimization for critical path testing using genetic algorithm. In J. Bansal, K. Das, A. Nagar, K. Deep, & A. Ojha (Eds.), *Soft computing for problem solving – Advances in intelligent systems and computing* (Vol. 817, pp. 67–80). Springer. https://doi.org/10.1007/978-981-13-1595-4_6

Mittal, S., & Sangwan, O. P. (2015). Metaheuristic based approach to regression testing. *International Journal of Computer Science and Information Technologies*, *6*(3), 2597–2605. http://ijcsit.com/docs/Volume 6/vol6issue03/ijcsit20150603135.pdf

Mohi-Aldeen, S. M., Mohamad, R., & Deris, S. (2016). Application of negative

selection algorithm (NSA) for test data generation of path Testing. *Applied Soft Computing Journal*, *49*, 1118–1128. https://doi.org/10.1016/j.asoc.2016.09.044

Mohi-Aldeen, S. M., Mohamad, R., & Deris, S. (2020). Optimal path test data generation based on hybrid negative selection algorithm and genetic algorithm. *PLoS ONE*, *15*(11 November), 1–21. https://doi.org/10.1371/journal.pone.0242812

Mustafa, A., Wan-Kadir, W. M. N., Ibrahim, N., Shah, M. A., Younas, M., Khan, A., Zareei, M., & Alanazi, F. (2021). Automated test case generation from requirements: A systematic literature review. *Computers, Materials and Continua*, *67*(2), 1819–1833. https://doi.org/10.32604/cmc.2021.014391

Nayyar, A., & Nguyen, N. G. (2018). Introduction to swarm intelligence. In A. Nayyar, D.-N. Le, & N. G. Nguyen (Eds.), *Advances in swarm intelligence for optimizing problems in computer science* (pp. 53–78). Chapman and Hall/CRC.

Nazir, M., & Khan, R. A. (2012). Testability estimation model (TEM OOD). In N. Meghanathan, N. Chaki, & D. Nagamalai (Eds.), *Advances in Computer Science and Information Technology. Computer Science and Information Technology. CCSIT 2012. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering* (Vol. 86, pp. 178–187). Springer. https://doi.org/10.1007/978-3-642-27317-9_19

Nidhra, S., & Dondeti, J. (2012). Black box and white box testing techniques - A literature review. *International Journal of Embedded Systems and Applications*, *2*(2), 29–50. https://doi.org/10.5121/ijesa.2012.2204

Nirpal, P. B., & Kale, K. V. (2011). Comparison of software test data for automatic

path coverage using genetic algorithm. *International Journal of Computer Science & Engineering Technology (IJCSET)*, *1*(1), 12–16. http://www.ijcset.com/docs/IJCSET10-01-01-03.pdf

Okezie, F., Odun-Ayo, I., & Bogle, S. (2019). A critical analysis of software testing tools. *Journal of Physics: Conference Series*, *1378*(4). https://doi.org/10.1088/1742-6596/1378/4/042030

Pachauri, A. (2011). Test data generation for unit testing of Java programs using JML and genetic algorithm. *International Journal of Software Engineering and Its Applications*, *1*(2), 11–20. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.675.5604&rep=rep1&type=pdf

Pachauri, A., & Srivastava, G. (2013). Automated test data generation for branch testing using genetic algorithm: An improved approach using branch ordering, memory and elitism. *Journal of Systems and Software*, *86*(5), 1191–1208. https://doi.org/10.1016/j.jss.2012.11.045

Parnami, S. (2013). Testing target path by automatic generation of test data using genetic algorithm. *International Journal of Information and Computation Technology*, *3*(8), 825–832. http://www.irphouse.com/ijict_spl/13_ijictv3n8spl.pdf

Poonam, S., & Tyagi, S. (2014). Test data generation for basis path testing using genetic algorithm and clonal selection algorithm. *International Journal of Science and Research (IJSR) ISSN*, *3*(6), 995–998. https://www.ijsr.net/archive/v3i6/MDIwMTQzODc=.pdf

Prajapati, V. K., Jain, M., & Chouhan, L. (2020). Tabu Search Algorithm (TSA): A comprehensive survey. *2020 3rd International Conference on Emerging Technologies in Computer Engineering: Machine Learning and Internet of Things (ICETCE)*, *May*, 222–229. https://doi.org/10.1109/ICETCE48199.2020.9091743

Prakash, B., Saleena, B., Shravan, S., & Vijayanandh, R. (2020). Optimization of test cases: A meta-heuristic approach. *International Journal of Advanced Trends in Computer Science and Engineering*, *9*(4), 6569–6576. https://doi.org/10.30534/ijatcse/2020/346942020

Rajakumar, B. R., & George, A. (2012). A new adaptive mutation technique for genetic algorithm. *2012 IEEE International Conference on Computational Intelligence and Computing Research (ICCIC)*, 1–7. https://doi.org/10.1109/ICCIC.2012.6510293

Rani, S., & Kaur, A. (2020). *A literature survey on automatic generation of test cases using genetic algorithm* (Issue July). https://www.researchgate.net/profile/Seema-Rani-14/publication/344263946_A_Literature_Survey_on_Automatic_Generation_of_Test_Cases_Using_Genetic_Algorithm/links/5f61b65692851c07896a44e7/A-Literature-Survey-on-Automatic-Generation-of-Test-Cases-Using-Genet

Ristić, O., Milošević, M., Radović, M., & Urošević, V. (2019). Genetic algorithm in software testing optimization : A review. *11th International Scientific Conference "Science and Higher Education in Function of Sustainable Development," May*, 61–66. http://www.vpts.edu.rs/sed/CD Proceedings 2019/proceedings/2-10.pdf

Sachdeva, T., & Pattanaik, A. (2020). Software test data generation based on path

testing using genetic algorithms. *International Journal of Recent Technology and Engineering (IJRTE)*, *8*(6), 4466–4473. https://doi.org/10.35940/ijrte.f9775.038620

Sahoo, R. R., & Ray, M. (2018). Metaheuristic techniques for test case generation: A review. *Journal of Information Technology Research*, *11*(1), 158–171. https://doi.org/10.4018/JITR.2018010110

Sakti, A., Pesant, G., & Guéhéneuc, Y. G. (2015). Instance generator and problem representation to improve object oriented code coverage. *IEEE Transactions on Software Engineering*, *41*(3), 294–313. https://doi.org/10.1109/TSE.2014.2363479

Salimans, T., Ho, J., Chen, X., Sidor, S., & Sutskever, I. (2017). Evolution strategies as a scalable alternative to reinforcement learning. In *arXiv:1703.03864*. https://doi.org/https://doi.org/10.48550/arXiv.1703.03864

Sangeetha, V., & Ramasundaram, T. (2016). Application of genetic algorithms in software testing techniques. *International Journal of Advanced Research in Computer and Communication Engineering*, *5*(10), 233–237. https://doi.org/10.17148/IJARCCE.2016.51047

Serdyukov, K., & Avdeenko, T. V. (2017). Investigation of the genetic algorithm possibilities for retrieving relevant cases from big data in the decision support systems. *3rd International Conference "Information Technology and Nanotechnology 2017," 1903*(January), 36–41. https://doi.org/10.18287/1613-0073-2017-1903-36-41

Serdyukov, K. S., & Avdeenko, T. V. (2018). Automatic data generation for software

testing based on the genetic algorithm. *2018 14th International Scientific-Technical Conference on Actual Problems of Electronic Instrument Engineering (APEIE)*, 535–540. https://doi.org/10.1109/APEIE.2018.8545975

Shah, H., Harrold, M. J., & Sinha, S. (2014). Global software testing under deadline pressure: Vendor-side experiences. *Information and Software Technology*, *56*(1), 6–19. https://doi.org/10.1016/j.infsof.2013.04.005

Shaheen, M., & Kosba, E. (2012). Software testing suite prioritization using multi-criteria fitness function. *2012 22nd International Conference on Computer Theory and Applications (ICCTA)*, *October*, 160–166. https://doi.org/10.1109/ICCTA.2012.6523563

Sharma, A., Patani, R., & Aggarwal, A. (2016). Software testing using genetic algorithms. *International Journal of Computer Science & Engineering Survey*, *7*(2), 21–33. https://doi.org/10.5121/ijcses.2016.7203

Sharma, S., & Bhatia, S. (2018). Comparative study of software testing through genetic algorithm. *International Journal of Computer Science and Information Security (IJCSIS)*, *16*(10), 27–33.

Sharma, S., & Bhatia, S. (2020). An algorithm for finding the optimal path in basis path testing using GABVIE Model. *International Journal of Innovative Technology and Exploring Engineering*, *9*(3), 587–593. https://doi.org/10.35940/ijitee.c8436.019320

Shimin, L., & Zhangang, W. (2011). Genetic algorithm and its application in the path-oriented test data automatic generation. *Procedia Engineering*, *15*, 1186–1190. https://doi.org/10.1016/j.proeng.2011.08.219

Shingadiya, C. J., & Sureja, N. M. (2021). Genetic algorithm for test suite optimization: An experimental investigation of different selection methods. *Turkish Journal of Computer and Mathematics Education (TURCOMAT)*, *12*(3), 3778–3787. https://doi.org/10.17762/turcomat.v12i3.1661

Shukla, A., Pandey, H. M., & Mehrotra, D. (2015). Comparative review of selection techniques in genetic algorithm. *2015 1st International Conference on Futuristic Trends in Computational Analysis and Knowledge Management (ABLAZE)*, *March*, 515–519. https://doi.org/10.1109/ABLAZE.2015.7154916

Silva, R. A., de Souza, S. do R. S., & de Souza, P. S. L. (2017). A systematic review on search based mutation testing. *Information and Software Technology*, *81*, 19–35. https://doi.org/10.1016/j.infsof.2016.01.017

Simos, D. E., Bozic, J., Garn, B., Leithner, M., Duan, F., Kleine, K., Lei, Y., & Wotawa, F. (2019). Testing TLS using planning-based combinatorial methods and execution framework. *Software Quality Journal*, *27*(2), 703–729. https://doi.org/10.1007/s11219-018-9412-z

Singh, G., Gupta, N., & Khosravy, M. (2016). New crossover operators for real coded genetic algorithm (RCGA). *ICIIBMS 2015 - International Conference on Intelligent Informatics and Biomedical Sciences*, 135–140. https://doi.org/10.1109/ICIIBMS.2015.7439507

Singh, I., & Tarika, B. (2014). Comparative analysis of open source automated software testing tools: Selenium, Sikuli and Watir. *International Journal of Information & Computation Technology*, *4*(15), 1507–1518. https://doi.org/10.13140/2.1.1418.4324

Srivastava, A. K., Khan, R., & Jain, S. (2019). A tool for generation of automatic control flow graph in unit testing of Python programs. *International Journal of Engineering and Advanced Technology*, *8*(5), 1178–1184. https://www.ijeat.org/wp-content/uploads/papers/v8i5/E7429068519.pdf

Srivastava, P. R., & Kim, T. (2009). Application of genetic algorithm in software testing. *International Journal of Software Engineering and Its Applications*, *3*(4), 87–96. https://d1wqtxts1xzle7.cloudfront.net/33594214/6-with-cover-page-v2.pdf?Expires=1655329971&Signature=dT7SD8bXCGZO28SaJ8VcHp9JDC83Yt3Qm4DXpASGd8RF4xjGYV-MhP-WCvlnzaV8qalCVWB6b-OcH0WeCYTwtEuTl7K1D4bjA0IWxwzQwGpZZj8cYK4IFhHcpUVpsKYMaPSY5uDuWyuLSiPujsZ~-f4pHM

Suresh, Y., & Rath, S. K. (2014). A genetic algorithm based approach for test data generation in basis path testing. *The International Journal of Soft Computing and Software Engineering (JSCSE)*, *3*(3), 326–332.

Tang, P.-H., & Tseng, M.-H. (2013). Adaptive directed mutation for real-coded genetic algorithms. *Applied Soft Computing*, *13*(1), 600–614. https://doi.org/10.1016/j.asoc.2012.08.035

Umar, M. A. (2019). Comprehensive study of software testing: Categories, levels, techniques, and types. *International Journal of Advance Research, Ideas and Innovations in Technology*, *5*(6), 32–40. https://doi.org/https://doi.org/10.36227/TECHRXIV.12578714.V2

Venkateswaran, C., Ramachandran, M., Chinnasamy, S., Sivaji, C., & Amudha, M. (2022). An extensive study on gravitational search algorithm. *Materials and Its*

Characterization, *1*(1), 9–16. https://doi.org/https://doi.org/10.46632/mc/1/1/2

Verma, A., Khatana, A., & Chaudhary, S. (2017). A comparative study of black box testing and white box testing. *International Journal of Computer Sciences and Engineering*, *5*(12), 301–304. https://doi.org/10.26438/ijcse/v5i12.301304

Vieira, F. E., Martins, F., Silva, R., Menezes, R., & Braga, M. (2006). Using genetic algorithms for test plans for functional testing. *Proceedings of the 44th Annual Southeast Regional Conference*, 140–145. https://doi.org/https://doi.org/10.1145/1185448.1185480

Vijay, N., & Vipin, K. K. S. (2017). Automated test path generation using genetic algorithm. *International Journal of Engineering Research And*, *V6*(07), 469–472. https://doi.org/10.17577/ijertv6is070262

Wang, R., Sato, Y., & Liu, S. (2021). Mutated specification-based test data generation with a genetic algorithm. *Mathematics*, *9*(4), 1–19. https://doi.org/10.3390/math9040331

Wegener, J., Baresel, A., & Sthamer, H. (2002). Suitability of evolutionary algorithms for evolutionary testing. *Proceedings 26th Annual International Computer Software and Applications*, 287–289. https://doi.org/10.1109/cmpsac.2002.1044566

Xibo, W., & Na, S. (2011). Automatic test data generation for path testing using genetic algorithms. *2011 Third International Conference on Measuring Technology and Mechatronics Automation*, 596–599. https://doi.org/10.1109/ICMTMA.2011.152

Yang, H. L., & Wang, C. S. (2008). Two stages of case-based reasoning – Integrating genetic algorithm with data mining mechanism. *Expert Systems with Applications*,

35(1–2), 262–272. https://doi.org/10.1016/j.eswa.2007.06.027

Yao, X., Gong, D., Li, B., Dang, X., & Zhang, G. (2020). Testing method for software with randomness using genetic algorithm. *IEEE Access*, *8*, 61999–62010. https://doi.org/10.1109/ACCESS.2020.2983762

Yao, X., Gong, D., & Wang, W. (2015). Test data generation for multiple paths based on local evolution. *Chinese Journal of Electronics*, *24*(1), 46–51. https://doi.org/10.1049/cje.2015.01.008

Zhu, Z., Xu, X., & Jiao, L. (2017). Improved evolutionary generation of test data for multiple paths in search-based software testing. *2017 IEEE Congress on Evolutionary Computation (CEC)*, 612–620. https://doi.org/10.1109/CEC.2017.7969367

Appendix A

The Simulation Program's Code

```
#include <iostream>
#include<bits/stdc++.h>
#include <string>
#include <stdlib.h>
#include <cstring>
#include <cstdlib>
using namespace std;
int v_start=1;  // the initial test data is generated randomly between
v_start and v_end (usually between 1 and 10)
int v_end=10;
const int size=5;
int parameter_1[size];     //These arrays hold the parameters (test
data)
int parameter_2[size];
int parameter_3[size];
int parameter_1_before[size]; //These arrays hold the test data before
crossover and repetitive mutation, in case the test data is not achieve
more than 20% path coverage rate it is not selected and the test data
before is selected.
int parameter_2_before[size];
int parameter_3_before[size];
int covered_path[5];  // this is the proposed array that holds the
paths that have not covered previously in the same generation (non-
duplicated)
void  initial_test_data_generation(int  v_start,  int  v_end,  int
experiment_no, int no_of_target_paths) // this function generates the
initial test data randomly between 1 and 10
{
    for(int i=1;i<=no_of_target_paths;i++)                        // 5
loops since we have 5 test data in the sequential SUT experiment
    {        static bool v_start = false;
             if (!v_start) {
                   srand(time(NULL));   // this function resets the
random value. without this code, after re execute the simulation
program it will take the same random value
                   v_start = true;
                 }
             int range = (v_end-v_start)+1;
             int  random_int  =  v_start+(rand()%range);   // this line
generates random number between v_start and v_end (1 and 10)
             parameter_1[i]=random_int;
             random_int  =  v_start+(rand()%range);      //this  line
generates random number between v_start and v_end (1 and 10)
             parameter_2[i]=random_int;
             if (experiment_no=1) // consider the three parameters
                    {
                      random_int = v_start+(rand()%range);
                      parameter_3[i]=random_int;
                     }
    }
```

```
}
int Check_Coverage(int Number_1, int Number_2, int Number_3, int
experiment_no)  // this function is to check the coverage of test data
on the specified SUT. The SUT is inside this function
{
  int greatest_number;
  int z;
  int LowerLimit, UpperLimit, i, Valid, j;
   int node1=0;
   int node2=0;
   int node3=0;
   int node4=0;
   int node5=0;

   switch (experiment_no)
    {
   case 1:

    //#include <iostream.h>             // Sequential SUT starts from
here
    //#include<stdio.h>
    //cout<<" Enter the values of a, b and c";
    //cin>>a>>b>>c;
       if(Number_1>Number_2)
            {if(Number_2>Number_3)
                {
                   greatest_number=Number_1;
               //           cout<<"The     greatest     number     is
"<<greatest_number<<endl;
                   node1=1;
                }
              else if(Number_1>Number_3)
                 {
                   greatest_number=Number_1;
                //        cout<<"The    greatest    number    is    "
<<greatest_number<<endl;
                    node2=1;
                   }
                else
                 {
                     greatest_number=Number_3;
                //           cout<<"    The    greatest    number    is
"<<greatest_number<<endl;
                     node3=1;

                  }
            }// first if
           else if(Number_2>Number_3)
                {
                   greatest_number=Number_2;
               //           cout<<"    The    greatest    number    is
"<<greatest_number<<endl;
                    node4=1;

                   }
              else
```

```
                    {
                     greatest_number=Number_3;
                //           cout<<"    The    greatest    number    is
"<<greatest_number<<endl;
                     node5=1;

                     }              // Sequential SUT ends here

              if (node1==1 and node2==0 and node3==0 and node4==0 and
node5==0)
                        return 1;
              else if(node1==0 and node2==1 and node3==0 and node4==0
and node5==0)
                        return 2;
              else if(node1==0 and node2==0 and node3==1 and node4==0
and node5==0)
                        return 3;
              else if(node1==0 and node2==0 and node3==0 and node4==1
and node5==0)
                        return 4;
              else if(node1==0 and node2==0 and node3==0 and node4==0
and node5==1)
                        return 5;
              else return 0;
    break;
  case 2:

        //#include <iostream.h> // Single loop SUT starts from here
        //#include<stdio.h>
        //cout <<" Enter the two numbers ";
        //  cin>>Number_1>>Number_2;
        if( Number_2 > Number_1 )
          {
            z = Number_1;
            Number_1 = Number_2;
            Number_2 =z;
            node1=1;
          }
          z = Number_1 % Number_2;
         while ( z != 0 )
           {
             Number_1 = Number_2;
             Number_2 = z;
             z = Number_1 % Number_2;
             node2=1;
            }
           //cout<<"GCD is :: "<<Number_2;   // Single loop SUT ends
here
            if (node1==1 and node2==1)
                      return 1;
            else if(node1==1 and node2==0)
                      return 2;
            else if(node1==0 and node2==1)
                      return 3;
            else if(node1==0 and node2==0)
                      return 4;
```

```
            else return 0;
    break;
  case 3:

        //#include <iostream.h> // Nested loops SUT starts from here
        //#include<stdio.h>
        //cout<<" Enter the values of Number_1 and Number_2";
        //cin>>Number_1>>Number_2;
        if (Number_2<Number_1)
          {
           int l;
           l=Number_1;
           Number_1=Number_2;
           Number_2=l;
         }
         LowerLimit=Number_1;
         UpperLimit=Number_2;
        //cout("\n Enter a range to find Prime Numbers: "); // SUT
starts from here
        //cin>>LowerLimit>>UpperLimit;
        //cout<<"Prime numbers in the range %d and %d are:
"<<LowerLimit<<UpperLimit;
        for(i=LowerLimit+1; i<UpperLimit; i++)
          {
              node1=1;
              Valid =0;
              for(j=2; j<=i/2; j++)
                {
                  node2=1;
                  if(i%j==0)
                   {
                    Valid=1;
                    node3=1;
                    break;
                   }
                 }//for
           if(Valid==0)
             {
              //Cout<<i;
               node4=1;
             }

   }//for
   //return 0;  // Nested loops SUT ends here

            if (node1==1 and node2==0 and node3==0 and node4==1)
                      return 1;
              else if(node1==1 and node2==1 and node3==0 and node4==1)
                       return 2;
              else if(node1==1 and node2==1 and node3==1 and node4==0)
                       return 3;
              else if(node1==1 and node2==1 and node3==1 and node4==1)
                       return 4;
              else if(node1==0 and node2==0 and node3==0 and node4==0)
                       return 5;
              //else return 0;
```

```
    break;
   }

}
string toBinary(int decimal)      // this function converts the decimal
test data to binary in order to perform the mutation and crossover on
the genes
{
    string str;
    string binary;
    while (decimal != 0){
        binary += ( decimal % 2 == 0 ? "0" : "1" );
        decimal /= 2;
    }
    reverse(binary.begin(), binary.end());
    int length = binary.length();
    if (8-length>0)
    {
        str=str.append(8-length,'0');
        binary=str.append(binary);
    }
    return binary;
}
int toDecimal(long binary)       // this function convert the binary
test data to real in order to return it to its original form
  {

     int decimalNum, i, div_remain;
     decimalNum = 0;
     i = 0;
     //converting binary to decimal
     while (binary != 0)
       {
          div_remain = binary % 10;
          binary /= 10;
          decimalNum += div_remain * pow(2, i);
          ++i;
   }
   return decimalNum;
}
void selection(float pathcount, int no_of_target_paths, int v_start,
int v_end, int experiment_no, float path_cov, int iteration_number)
 {
    int path1=0;   //these ,, path1 to path 5 variables are used to
check the coverage later
    int path2=0;
    int path3=0;
    int path4=0;
    int path5=0;
    int coveredpath=0;
    float path_cov_previous;
   if (iteration_number==1)
   {
     while ((pathcount/no_of_target_paths)<=0.2)
      {
```

```
          initial_test_data_generation(v_start,v_end,  experiment_no,
no_of_target_paths);
          cout<<endl;
          cout<<"  The Initial Selected Test Data are:: "<<endl;;
          cout<<endl;

    for(int i=1; i<=no_of_target_paths;i++)  // Printing the initial
population. 5 loops since we have 5 test data
       {
               cout<<endl;
               cout<<"  ["<<parameter_1[i]<<"]"<<" ";
               cout<<"["<<parameter_2[i]<<"]"<<" ";
               if (experiment_no==1)
                {
                 cout<<"["<<parameter_3[i]<<"]"<<" ";
                 cout<<endl;
               }
               coveredpath=Check_Coverage(parameter_1[i],
parameter_2[i], parameter_3[i], experiment_no); // the initial test
data coverage checking
                cout<<"        The    Covered    Path    is::    Path
"<<coveredpath<<endl;

cout<<"_____________________________________________"<<endl;

                if (coveredpath==1)
                       path1=1;
                else if (coveredpath==2)
                       path2=1;
                else if (coveredpath==3)
                       path3=1;
                else if (coveredpath==4)
                       path4=1;
                else if (coveredpath==5)
                       path5=1;
       }
       path1=0;   //reset thee variables to hold new values
       path2=0;
       path3=0;
       path4=0;
       path5=0;
       coveredpath=0;
       if (experiment_no==1 or experiment_no==3)
       {
           pathcount=path1+path2+path3+path4+path5;
           cout<<endl;
           cout<<"       "<<pathcount<<"       paths       out       of
"<<no_of_target_paths<<" paths have been covered and the coverage rate=
"<<((pathcount/no_of_target_paths)*100)<<"%"<<endl;

       }
      else
      {
         pathcount=path1+path2+path3+path4;
         cout<<endl;
```

```
          cout<<"      "<<pathcount<<"      paths      out      of
"<<no_of_target_paths<<" paths have been covered and the coverage rate=
"<<((pathcount/no_of_target_paths)*100)<<"%"<<endl;
      }


    }// while
   }
     else
   {  coveredpath=0;
       pathcount=0;
       path1=0;   //reset thee variables to hold new values
       path2=0;
       path3=0;
       path4=0;
       path5=0;
       path_cov_previous=0;
       for(int  i=1;  i<=no_of_target_paths;i++)    //  Printing  the
initial population. 5 loops since we have 5 test data
       {
                coveredpath=Check_Coverage(parameter_1_before[i],
parameter_2_before[i], parameter_3_before[i], experiment_no);
                  if (coveredpath==1)
                         path1=1;
                  else if (coveredpath==2)
                         path2=1;
                  else if (coveredpath==3)
                         path3=1;
                  else if (coveredpath==4)
                         path4=1;
                  else if (coveredpath==5)
                         path5=1;
       }

       if (experiment_no==1 or experiment_no==3)
       {
           pathcount=path1+path2+path3+path4+path5;
           path_cov_previous=(pathcount/no_of_target_paths)*100;
       }
      else
      {
         //cout<<"pathcount= "<<pathcount<<endl;
         pathcount=path1+path2+path3+path4;
           path_cov_previous=(pathcount/no_of_target_paths)*100;
      }
      cout<<"path_cov= "<<path_cov<<endl;
      cout<<"path_cov_previous= "<<path_cov_previous<<endl;
      cout<<"path Count Previous ="<<pathcount<<endl;
       cout<<"path1 ="<<path1<<endl;
       cout<<"path2 ="<<path2<<endl;
       cout<<"path3 ="<<path3<<endl;
       cout<<"path4 ="<<path4<<endl;
        cout<<"path5 ="<<path5<<endl;
        for(int i=1; i<=no_of_target_paths;i++)
            {
                    cout<<parameter_1_before[i]<<" ";
```

```
                    cout<<parameter_2_before[i]<<endl;

            }
    if (path_cov < path_cov_previous)
      {
         for(int i=1; i<=no_of_target_paths;i++)
            {
                    parameter_1[i]=parameter_1_before[i];
                    parameter_2[i]=parameter_2_before[i];
                    parameter_3[i]=parameter_3_before[i];
            }
    }
   }

cout<<"=============================================================
======================"<<endl;
     cout<<endl;
}
void  crossover(int  parameter_1_befor,  int  parameter_2_befor,  int
parameter_3_befor,    string    &    parameter_1_after,    string    &
parameter_2_after , string & parameter_3_after, int experiment_no)  //
this function perform the crossover operation on the test data
{    string temp;
     int pos;

        if (experiment_no==1)
            {
              parameter_1_after= toBinary(parameter_1_befor);
              parameter_2_after= toBinary(parameter_2_befor);
              parameter_3_after= toBinary(parameter_3_befor);
             temp={};
              pos = rand()% 8 + 1;   // random number between 1 and 8
              while (pos>=7) // to remain the crossover in the range
of 8 genes
              {
                pos = rand()% 8 + 1;
              }
              cout<<"     The Points for crossover between parameter
1 and 2 were :: "<<pos<<" , "<<pos+1<<" , "<<pos+2<<endl;
              temp=parameter_1_after;
              parameter_1_after.replace(pos-
1,3,(parameter_2_after.substr(pos-1,3)));
              parameter_2_after.replace(pos-1,3,(temp.substr(pos-
1,3)));
              pos = rand()% 8 + 1;   // random number between 1 and 8
               while (pos>=7) // to remain the crossover in the range
of 8 genes
              {
                pos = rand()% 8 + 1;
              }
              cout<<"     The Points for crossover between parameter
2 and 3 were:: "<<pos<<" , "<<pos+1<<" , "<<pos+2<<endl;
              temp=parameter_2_after;
              parameter_2_after.replace(pos-
1,3,(parameter_3_after.substr(pos-1,3)));
```

```
              parameter_3_after.replace(pos-1,3,(temp.substr(pos-
1,3)));
            }
          else
            {
              parameter_1_after= toBinary(parameter_1_befor);
              parameter_2_after= toBinary(parameter_2_befor);
              temp={};
              pos = rand()% 8 + 1;   // random number between 1 and 8
              while (pos>=7) // to remain the crossover in the range
of 8 genes
              {
                pos = rand()% 8 + 1;
              }
              cout<<"     The Points for crossover were :: "<<pos<<"
, "<<pos+1<<" , "<<pos+2<<endl;
              temp=parameter_1_after;
              parameter_1_after.replace(pos-
1,3,(parameter_2_after.substr(pos-1,3)));
              parameter_2_after.replace(pos-1,3,(temp.substr(pos-
1,3)));

            }
                if (experiment_no==2)
                 {
                    while      (parameter_1_after=="00000000"      or
parameter_2_after=="00000000") //zero makes problems in the SUTs if a
number divide on it.
                    {
                          parameter_1_after=
toBinary(parameter_1_befor);
                          parameter_2_after=
toBinary(parameter_2_befor);
                         temp={};
                         temp=parameter_1_after;
                          pos = rand()% 8 + 1;   // random number between
1 and 8
                          while (pos>=7) // to remain the crossover in
the range of 8 genes
                           {
                              pos = rand()% 8 + 1;
                            }
                         temp=parameter_1_after;
                         parameter_1_after.replace(pos-
1,3,(parameter_2_after.substr(pos-1,3)));
                         parameter_2_after.replace(pos-
1,3,(temp.substr(pos-1,3)));
                      }
                  }


          }
void  mutation(int  parameter_1_befor,  int  parameter_2_befor,  int
parameter_3_befor,    string    &    parameter_1_after,    string    &
parameter_2_after , string & parameter_3_after, int experiment_no) //
```

```
this function perform the repetitive mutation operation on the test
data
{
     int pos=0;

              parameter_1_after= toBinary(parameter_1_befor);
              parameter_2_after= toBinary(parameter_2_befor);
              parameter_3_after= toBinary(parameter_3_befor);
              pos = rand()% 8 + 1;   /* random number between 1 and 8
since our chromosome consist of 8 genes, so we have to choose random
position between
                                        1 and 8 */
              if (parameter_1_after.substr(pos-1,1)=="0")
                      parameter_1_after.replace(pos-1,1,"1");
              else if (parameter_1_after.substr(pos-1,1)=="1")
                       parameter_1_after.replace(pos-1,1,"0");


             pos = rand()% 8 + 1;   // random number between 1 and 8
              if (parameter_2_after.substr(pos-1,1)=="0")
                       parameter_2_after.replace(pos-1,1,"1");
              else if (parameter_2_after.substr(pos-1,1)=="1")
                         parameter_2_after.replace(pos-1,1,"0");


             pos = rand()% 8 + 1;   // random number between 1 and 8
              if (parameter_3_after.substr(pos-1,1)=="0")
                      parameter_3_after.replace(pos-1,1,"1");
              else if (parameter_3_after.substr(pos-1,1)=="1")
                      parameter_3_after.replace(pos-1,1,"0");

               if (experiment_no==2)
               {
                    while      (parameter_1_after=="00000000"      or
parameter_2_after=="00000000") //  zero makes problems in the SUTs if
a number divide on it.
                      {
                          parameter_1_after=
toBinary(parameter_1_befor);
                          parameter_2_after=
toBinary(parameter_2_befor);
                          pos = rand()% 8 + 1;
                         if (parameter_1_after.substr(pos-1,1)=="0")
                              parameter_1_after.replace(pos-1,1,"1");
                         else    if    (parameter_1_after.substr(pos-
1,1)=="1")
                             parameter_1_after.replace(pos-1,1,"0");
                         pos = rand()% 8 + 1;
                         if (parameter_2_after.substr(pos-1,1)=="0")
                             parameter_2_after.replace(pos-1,1,"1");
                          else    if    (parameter_2_after.substr(pos-
1,1)=="1")
                              parameter_2_after.replace(pos-1,1,"0");
                        }
                }
```

```
}
void   improved_repetitive_mutation(int   i,   int   cdc,   int   &
covered_path_index, int & coveredpath , int experiment_no, float
no_of_target_paths )
{
    string parameter_1_after;
    string parameter_2_after;
    string parameter_3_after;
    int mutation_position;
    int xb; // these variables hold the test data before the mutation
, if CDC exceeded without elimination path coverage, then return to
the original test data
    int yb; // same
    int zb; //same
    int flag=1;
    int found=0;
    float chromosome_rank;
    int mutation_number=0;
              xb=parameter_1[i];
              yb=parameter_2[i];
              zb=parameter_3[i];
              mutation(parameter_1[i],parameter_2[i],parameter_3[i],
parameter_1_after,        parameter_2_after,        parameter_3_after,
experiment_no); // Perform mutation on the test data
              parameter_1[i]=toDecimal(stol(parameter_1_after));
// convert the test data to decimal representation
              parameter_2[i]=toDecimal(stol(parameter_2_after));
// same
              cout<<endl;

               if (experiment_no==1) // if to print test data for 3
parameters EXPR
                  cout<<"  After Normal Single Mutation on the test
data"<<"  ["<<xb<<"]"<<"  ["<<yb<<"]"<<"  ["<<zb<<"]"<<" by flipping
one gene randomly, The produced test data IS::"<<endl;
                else // if to print test data for 3 parameters EXPR
                   cout<<"  After Normal Single Mutation on the test
data"<<"  ["<<xb<<"]"<<"  ["<<yb<<"]"<<" by flipping one gene randomly,
The produced test data IS::"<<endl;
               cout<<"   ["<<parameter_1[i]<<"]"<<" ";   // Print the
new test data after the affection of mutation technique
               cout<<"["<<parameter_2[i]<<"]"<<" ";
              if (experiment_no==1) // if to print test data for 3
parameters EXPR
              {
                   parameter_3[i]=toDecimal(stol(parameter_3_after));
                   cout<<"["<<parameter_3[i]<<"]"<<" ";
                   cout<<endl;
              }
              coveredpath=Check_Coverage(parameter_1[i],
parameter_2[i], parameter_3[i], experiment_no); //Check the coverage
of the new test data
              while (flag==1)  // while the new covered path is found
in the array ( Path Coverage Duplication), and CDC is not exceeded, do
                  {
```

```
                    for (int j=1;j<=5;j++) // Search in the array on
the covered path of the new test data
                     {
                      if (covered_path[j]==coveredpath) //   if the
covered path by that test data exists in the array
                           found=1;
                      }
                    if (found==1) // if the covered path by the new
test data is found in the array (path coverage duplication)
                     {

chromosome_rank=1/(1+mutation_number+no_of_target_paths);
                           mutation_number++;
                           cout<<"  This Test Data covers duplicated
path     Which     is::     Path     "<<coveredpath<<"     ,     Rank=
"<<chromosome_rank<<endl;
                           cout<<endl;
                           mutation(xb,yb,zb,      parameter_1_after,
parameter_2_after, parameter_3_after,experiment_no ); // Redo mutation
on the test data

parameter_1[i]=toDecimal(stol(parameter_1_after));

parameter_2[i]=toDecimal(stol(parameter_2_after));
                           cout<<"          After     Mutation     (
"<<mutation_number<<" ) on the same test data by flipping one gene
randomly, the new test data IS::"<<endl; //Print the mutation number
// optional
                           cout<<"    ["<<parameter_1[i]<<"]"<<"   ";
//Print the new test data after the affection of mutation technique //
optional
                           cout<<"["<<parameter_2[i]<<"]"<<" ";
                           if (experiment_no==1)  // if to print test
data for 3 parameters EXPR
                             {

parameter_3[i]=toDecimal(stol(parameter_3_after));
                                 cout<<"["<<parameter_3[i]<<"]"<<" ";
                                 cout<<endl;
                             }

coveredpath=Check_Coverage(parameter_1[i],            parameter_2[i],
parameter_3[i], experiment_no); // Check the coverage of the new test
data to see which path it covers
                           }
                         else //if not found in the array// means there
is no path coverage duplication
                           {
                              cout<<endl;
                              cout<<"   ["<<parameter_1[i]<<"]"<<"  ";
//Print the test data
                              cout<<"["<<parameter_2[i]<<"]"<<" ";
                              if (experiment_no==1)
                                 cout<<"["<<parameter_3[i]<<"]"<<" ";
                                 cout<<endl;
```

```
coveredpath=Check_Coverage(parameter_1[i],            parameter_2[i],
parameter_3[i], experiment_no); //Check the coverage of the new test
data
                              chromosome_rank=1/no_of_target_paths;
                               cout<<"   This Test Data covers non-
duplicated path which is:: Path "<<coveredpath<<" , Rank=
"<<chromosome_rank<<endl; //Print the covered path that is not
duplicated
                               cout<<endl;

covered_path[covered_path_index]=coveredpath; //Add the covered path
to the array
                               covered_path_index+=1;   //Increase the
array's index
                               flag=0;
                             }
                    if (mutation_number>=cdc and found==1)   // If
there is a path coverage duplication but the CDC is  exceeded
                           { mutation_number++;

chromosome_rank=1/(1+mutation_number+no_of_target_paths);
                            cout<<"  This Test Data covers duplicated
path     Which     is::     Path     "<<coveredpath<<"     ,     Rank=
"<<chromosome_rank<<endl;
                            cout<<endl;
                            parameter_1[i]=xb;      //ignore the new
test data that causes path coverage duplication and return to the the
original one
                            parameter_2[i]=yb;
                            if (experiment_no==1)
                              parameter_3[i]=zb;
                            mutation_number=0;
                            flag=0; // Make the while ends (set the
flag to zero to exist from the while loop since CDC has been exceeded)
                            }
                              found=0;
                   }
                   mutation_number=0;

                //after that , Select the next test data (in the main
by the for loop)
}
int main()
{

//_____________________________________________________________________
__ introductions
    int experiment_no;
    float no_of_target_paths;
    int coveredpath;
    int cdc=0;
    int path1=0;   //these ,, path1 to path 5 variables are used to
check the coverage later
    int path2=0;
    int path3=0;
```

```
    int path4=0;
    int path5=0;
    string x;
    string y;
    string z;
    int xb;
    int yb;
    int zb;
    int g=1;
    float pathcount=0;
    float path_cov;
    int iteration_number=0;
    int flag;
 //-
_____________________________________________________________________________End
d of introductions
    cout<<endl;
    cout<<"Enter 1 For Sequential Experiment "<<endl;
    cout<<"Enter 2 For Single Loop Experiment "<<endl;
    cout<<"Enter 3 For Nested Loops Experiment "<<endl;
    cin>>experiment_no;
    if (experiment_no ==2)
       no_of_target_paths=4;
    else
    {
       no_of_target_paths=5;
    }
    cout<<"Enter CDC Value"<<endl;
    cin>>cdc;

cout<<"_____________________________________________________________________
_"<<endl;
    initial_test_data_generation(v_start,v_end,    experiment_no,
no_of_target_paths); // this procedure generates the initial test data
    cout<<endl;
    cout<<"  The Initial Selected Test Data are:: "<<endl;;
    cout<<endl;
    for(int i=1; i<=no_of_target_paths;i++)
                        {
                                cout<<endl;
                cout<<"  ["<<parameter_1[i]<<"]"<<" ";
                cout<<"["<<parameter_2[i]<<"]"<<" ";
                if (experiment_no==1)
                 {
                  cout<<"["<<parameter_3[i]<<"]"<<" ";
                }
                coveredpath=Check_Coverage(parameter_1[i],
parameter_2[i], parameter_3[i], experiment_no); // the initial test
data coverage checking
                 cout<<"----->    The    Covered    Path    is::    Path
"<<coveredpath<<endl;

cout<<"_____________________________________________________"<<endl;

                 if (coveredpath==1)
                        path1=1;
```

```
                else if (coveredpath==2)
                       path2=1;
                else if (coveredpath==3)
                       path3=1;
                else if (coveredpath==4)
                       path4=1;
                else if (coveredpath==5)
                       path5=1;
        }
        if (experiment_no==1 or experiment_no==3)
        {
            pathcount=path1+path2+path3+path4+path5;
            cout<<endl;
            cout<<"       "<<pathcount<<"       paths       out       of
"<<no_of_target_paths<<" paths have been covered and the coverage rate=
"<<((pathcount/no_of_target_paths)*100)<<"%"<<endl;

        }
       else
       {
          pathcount=path1+path2+path3+path4;
          cout<<endl;
           cout<<"       "<<pathcount<<"       paths       out       of
"<<no_of_target_paths<<" paths have been covered and the coverage rate=
"<<((pathcount/no_of_target_paths)*100)<<"%"<<endl;
       }

   while (pathcount!=no_of_target_paths)  // this is the main loop and
it does not stop till all target paths covered
     {
           path1=0;
           path2=0;
           path3=0;
           path4=0;
           path5=0;
           cout<<endl;
           iteration_number+=1;
           selection(pathcount,  no_of_target_paths,  v_start,  v_end  ,
experiment_no , path_cov, iteration_number );
            cout<<endl;
            cout<<"  Iteration "<<iteration_number<<endl;   // printing
the iteration number
           cout<<"__________________"<<endl<<endl;
           if (iteration_number>1)
           {
           cout<<"  The Selected Test Data are:: "<<endl;
           for(int i=1; i<=no_of_target_paths;i++)
                      {
                                 cout<<endl;
               cout<<"  ["<<parameter_1[i]<<"]"<<" ";
               cout<<"["<<parameter_2[i]<<"]"<<" ";
               if (experiment_no==1)
                {
                 cout<<"["<<parameter_3[i]<<"]"<<" ";
                 cout<<endl;
               }
```

```
                    }
                cout<<endl;

cout<<"_____________________________________________"<<endl;
           }
           //srand(time(NULL));
        for(int i=1; i<=no_of_target_paths;i++) //------------------
-------------perform repetitive Mutation technique / on the 5 test data
same generation
           {
            parameter_1_before[i]=parameter_1[i]; // holds the test
data before it changes during the repetitive mutation for selection
purposes
            parameter_2_before[i]=parameter_2[i];
            parameter_3_before[i]=parameter_3[i];
           }
          float r;
          r= ((double) rand() / (RAND_MAX));  // find random number
between 0 and 1 for crossover probability
          if (r<=0.8) // 0.8 is the probability of crossover
          {
            cout<<endl;
            // Test Data Before Crossover
            cout<<"  The Crossover Operation :"<<endl;
            cout<<"  _________________________"<<endl<<endl;
            cout<<"  Test Data Before Crossover ::"<<endl;
            cout<<endl;
             for(int c=1; c<=no_of_target_paths;c++) //--------------
------------------perform crossover on the 5 test data
              {


                cout<<"          ["<<parameter_1[c]<<"]"<<"  ";   //
printing the test data after crossover operation
                cout<<"["<<parameter_2[c]<<"]"<<" ";
                if (experiment_no==1)
                cout<<"["<<parameter_3[c]<<"]"<<" ";
                 cout<<" Which equals ";
                cout<<"  ["<<toBinary(parameter_1[c])<<"]";
                cout<<"["<<toBinary(parameter_2[c])<<"]";
                if (experiment_no==1)
                cout<<"["<<toBinary(parameter_3[c])<<"]";
                cout<<" in binary";
                cout<<endl<<endl;

                }

            //Test Data After Crossover
            cout<<endl;
            cout<<"  Test Data After Crossover ::"<<endl;
            cout<<endl;
             for(int c=1; c<=no_of_target_paths;c++) //--------------
------------------perform crossover on the 5 test data
               {
```

```
crossover(parameter_1[c],parameter_2[c],parameter_3[c],     x,     y,
z,experiment_no);  // perform crossover function
                   parameter_1[c]=toDecimal(stol(x));   //   crossover
returns binary test data , so its must be convert to real test data
before printing them
                   parameter_2[c]=toDecimal(stol(y));
                   if (experiment_no==1)
                   parameter_3[c]=toDecimal(stol(z));
                    cout<<"        ["<<parameter_1[c]<<"]"<<" ";   //
printing the test data after crossover operation
                   cout<<"["<<parameter_2[c]<<"]"<<" ";
                   if (experiment_no==1)
                   cout<<"["<<parameter_3[c]<<"]"<<" ";
                   cout<<" Which equals ";
                   cout<<"  ["<<toBinary(parameter_1[c])<<"]";
                   cout<<"["<<toBinary(parameter_2[c])<<"]";
                   if (experiment_no==1)
                   cout<<"["<<toBinary(parameter_3[c])<<"]";
                   cout<<" in binary";
                   cout<<endl<<endl;

                   }
 cout<<"_____Crossover    has
finished :"<<endl;

                }
                else
                {
                   cout<<endl;
                   cout<<" (The probability of crossover has not
achieved) "<<endl;
                }
                r =0.0;
                r = ((double) rand() / (RAND_MAX));  // find random
number between 0 and 1

                   if (r<=0.8)// 0.8 is the probability of mutation
                   { int  coveredpath;
                     cout<<endl;
                     cout<<"     The   Proposed   Repetitive   Mutation
Operation :"<<endl;
                     cout<<"
_____"<<endl;
                for(int i=1; i<=no_of_target_paths;i++) //----------
----------perform Repetitive Mutation technique / on the 5
test data same generation
                {
                  improved_repetitive_mutation(i,cdc,g,coveredpath,
experiment_no, no_of_target_paths);
                     if (coveredpath==1)
                        path1=1;
                   else if (coveredpath==2)
                        path2=1;
                   else if (coveredpath==3)
                        path3=1;
```

```
                    else if (coveredpath==4)
                         path4=1;
                    else if (coveredpath==5)
                         path5=1;

                    }

                  }
                  else
                  {
                     cout<<endl;
                     cout<<" (The probability of mutation has not
achieved) "<<endl<<endl;
                     for(int i=1; i<=no_of_target_paths;i++) //-------
-------------------------perform Repetitive Mutation technique / on
the 5 test data same generation
                        {
                             cout<<endl;
                            cout<<"  ["<<parameter_1[i]<<"]"<<" ";
                            cout<<"["<<parameter_2[i]<<"]"<<" ";
                            if (experiment_no==1)
                               {
                                cout<<"["<<parameter_3[i]<<"]"<<" ";
                               }
                           coveredpath=Check_Coverage(parameter_1[i],
parameter_2[i], parameter_3[i], experiment_no); // the initial test
data coverage checking
                           cout<<" -----> The Covered Path is:: Path
"<<coveredpath<<endl;

cout<<"_____"<<endl;
                           cout<<endl;
                           if (coveredpath==1)
                               path1=1;
                           else if (coveredpath==2)
                               path2=1;
                           else if (coveredpath==3)
                               path3=1;
                           else if (coveredpath==4)
                               path4=1;
                           else if (coveredpath==5)
                               path5=1;
                         }
                   }

                  for (int m=1; m<=no_of_target_paths; m++)
                  {
                    covered_path[m]=0;   // reset the array to a new
iteration
                  }
                   g=1;
                    if (experiment_no==1 or experiment_no==3)
                     {
                    pathcount=path1+path2+path3+path4+path5;
                    cout<<endl;
```

```
                    cout<<"     "<<pathcount<<"     paths     out     of
"<<no_of_target_paths<<" paths have been covered,";
                    path_cov=(pathcount/no_of_target_paths)*100;
                    cout<<"  Path coverage rate= "<<path_cov<<"%"<<endl;
// the fittest test data coverage
                    cout<<endl;
                     }
                    else
                    {
                       pathcount=path1+path2+path3+path4;
                    cout<<endl;
                    cout<<"     "<<pathcount<<"     paths     out     of
"<<no_of_target_paths<<" paths have been covered,";
                    path_cov=(pathcount/no_of_target_paths)*100;
                    cout<<"  Path coverage rate= "<<path_cov<<"%"<<endl;
// the fittest test data coverage
                    cout<<endl;
                     }
                    cout<<"   The  number  of  test  data  iterations  is
"<<iteration_number<<endl;

cout<<"_____________________________________________________________________
_"<<endl<<endl;
                     path1=0;   //these ,, path1 to path 5 variables are
used to check the coverage later
                     path2=0;
                     path3=0;
                     path4=0;
                     path5=0;

          }
      cout<<" the fittest test data for path coverage testing are::
"<<endl<<endl; //printing the final fittest test data that cover all
paths

            for(int i=1; i<=no_of_target_paths;i++)
               {
                   cout<<"  ["<<parameter_1[i]<<"]"<<" ";
                   cout<<"["<<parameter_2[i]<<"]"<<" ";
                   if (experiment_no==1)
                       cout<<"["<<parameter_3[i]<<"]"<<" ";
                       cout<<"----->  The  Covered  Path  is::  Path
"<<Check_Coverage(parameter_1[i],   parameter_2[i],   parameter_3[i],
experiment_no);

cout<<endl<<"_____________________________________________________"<<endl;
                       cout<<endl;

                }

}
```

Appendix B

The Results' Snapshots of Sequential SUT Experiment

```
Enter 1 For Sequential Experiment
Enter 2 For Single Loop Experiment
Enter 3 For Nested Loops Experiment
1
Enter CDC Value
5
```

The Simulation Program's Inputs

```
The Initial Selected Test Data are::

[6] [8] [2] -----> The Covered Path is:: Path 4
_____________________________________

[4] [10] [9] -----> The Covered Path is:: Path 4
_____________________________________

[9] [10] [9] -----> The Covered Path is:: Path 4
_____________________________________

[3] [4] [10] -----> The Covered Path is:: Path 5
_____________________________________

[9] [10] [9] -----> The Covered Path is:: Path 4
_____________________________________

2 paths out of 5 paths have been covered and the coverage rate= 40%
```

The Initial Test Data

```
Iteration 1
_____________


The Crossover Operation :
_________________________

Test Data Before Crossover ::

    [6] [8] [2]  Which equals   [00000110][00001000][00000010] in binary

    [4] [10] [9]  Which equals   [00000100][00001010][00001001] in binary

    [9] [10] [9]  Which equals   [00001001][00001010][00001001] in binary

    [3] [4] [10]  Which equals   [00000011][00000100][00001010] in binary

    [9] [10] [9]  Which equals   [00001001][00001010][00001001] in binary


Test Data After Crossover ::

   The Points for crossover between parameter 1 and 2 were :: 4 , 5 , 6
   The Points for crossover between parameter 2 and 3 were:: 3 , 4 , 5
    [10] [4] [2]  Which equals   [00001010][00000100][00000010] in binary

   The Points for crossover between parameter 1 and 2 were :: 6 , 7 , 8
   The Points for crossover between parameter 2 and 3 were:: 4 , 5 , 6
    [2] [8] [13]  Which equals   [00000010][00001000][00001101] in binary

   The Points for crossover between parameter 1 and 2 were :: 3 , 4 , 5
   The Points for crossover between parameter 2 and 3 were:: 5 , 6 , 7
    [9] [8] [11]  Which equals   [00001001][00001000][00001011] in binary

   The Points for crossover between parameter 1 and 2 were :: 1 , 2 , 3
   The Points for crossover between parameter 2 and 3 were:: 1 , 2 , 3
    [3] [4] [10]  Which equals   [00000011][00000100][00001010] in binary

   The Points for crossover between parameter 1 and 2 were :: 4 , 5 , 6
   The Points for crossover between parameter 2 and 3 were:: 6 , 7 , 8
    [9] [9] [10]  Which equals   [00001001][00001001][00001010] in binary

                                                Crossover has finished :
```

Iteration 1 - Crossover

```
The Proposed Multi Mutation Operation :
_______________________________________

After Normal Single Mutation on the test data  [10]  [4]  [2] by flipping one gene randomly, The produced test data IS::
[42] [5] [34]

[42] [5] [34]
This Test Data covers non-duplicated path which is:: Path 2 , Rank= 0.2


After Normal Single Mutation on the test data  [2]  [8]  [13] by flipping one gene randomly, The produced test data IS::
[3] [72] [15]

[3] [72] [15]
This Test Data covers non-duplicated path which is:: Path 4 , Rank= 0.2


After Normal Single Mutation on the test data  [9]  [8]  [11] by flipping one gene randomly, The produced test data IS::
[41] [0] [75]

[41] [0] [75]
This Test Data covers non-duplicated path which is:: Path 3 , Rank= 0.2
```

Iteration 1 - The Proposed Repetitive Mutation -Part1-

```
After Normal Single Mutation on the test data  [3]  [4]  [10] by flipping one gene randomly, The produced test data IS::
[11] [12] [11]
This Test Data covers duplicated path Which is:: Path 4 , Rank= 0.166667

After Mutation ( 1 ) on the same test data by flipping one gene randomly, the new test data IS::
[7] [20] [11]
This Test Data covers duplicated path Which is:: Path 4 , Rank= 0.142857

After Mutation ( 2 ) on the same test data by flipping one gene randomly, the new test data IS::
[1] [132] [74]
This Test Data covers duplicated path Which is:: Path 4 , Rank= 0.125

After Mutation ( 3 ) on the same test data by flipping one gene randomly, the new test data IS::
[67] [5] [42]
This Test Data covers duplicated path Which is:: Path 2 , Rank= 0.111111

After Mutation ( 4 ) on the same test data by flipping one gene randomly, the new test data IS::
[11] [12] [74]

[11] [12] [74]
This Test Data covers non-duplicated path which is:: Path 5 , Rank= 0.2
```

Iteration 1 - The Proposed Repetitive Mutation -Part2-

```
After Normal Single Mutation on the test data  [9]  [9]  [10] by flipping one gene randomly, The produced test data IS::
[8] [25] [8]
This Test Data covers duplicated path Which is:: Path 4 , Rank= 0.166667

After Mutation ( 1 ) on the same test data by flipping one gene randomly, the new test data IS::
[11] [8] [8]
This Test Data covers duplicated path Which is:: Path 2 , Rank= 0.142857

After Mutation ( 2 ) on the same test data by flipping one gene randomly, the new test data IS::
[137] [1] [74]
This Test Data covers duplicated path Which is:: Path 2 , Rank= 0.125

After Mutation ( 3 ) on the same test data by flipping one gene randomly, the new test data IS::
[73] [73] [14]
This Test Data covers duplicated path Which is:: Path 4 , Rank= 0.111111

After Mutation ( 4 ) on the same test data by flipping one gene randomly, the new test data IS::
[73] [1] [26]
This Test Data covers duplicated path Which is:: Path 2 , Rank= 0.1

After Mutation ( 5 ) on the same test data by flipping one gene randomly, the new test data IS::
[25] [1] [11]
This Test Data covers duplicated path Which is:: Path 2 , Rank= 0.0833333


4 paths out of 5 paths have been covered,  Path coverage rate= 80%

The number of test data iterations is 1
```

Iteration 1 - The Proposed Repetitive Mutation -Part3-

```
Iteration 2
_____________

The Selected Test Data are::

[42] [5] [34]

[3] [72] [15]

[41] [0] [75]

[11] [12] [74]

[9] [9] [10]
```

Iteration 2 - The Selected Test Data

```
The Crossover Operation :
_____________________

Test Data Before Crossover ::

    [42] [5] [34]  Which equals   [00101010][00000101][00100010] in binary

    [3] [72] [15]  Which equals   [00000011][01001000][00001111] in binary

    [41] [0] [75]  Which equals   [00101001][00000000][01001011] in binary

    [11] [12] [74]  Which equals   [00001011][00001100][01001010] in binary

    [9] [9] [10]  Which equals   [00001001][00001001][00001010] in binary


Test Data After Crossover ::

   The Points for crossover between parameter 1 and 2 were :: 1 , 2 , 3
   The Points for crossover between parameter 2 and 3 were:: 2 , 3 , 4
    [10] [37] [34]  Which equals   [00001010][00100101][00100010] in binary

   The Points for crossover between parameter 1 and 2 were :: 6 , 7 , 8
   The Points for crossover between parameter 2 and 3 were:: 6 , 7 , 8
    [0] [79] [11]  Which equals   [00000000][01001111][00001011] in binary

   The Points for crossover between parameter 1 and 2 were :: 6 , 7 , 8
   The Points for crossover between parameter 2 and 3 were:: 4 , 5 , 6
    [40] [9] [67]  Which equals   [00101000][00001001][01000011] in binary

   The Points for crossover between parameter 1 and 2 were :: 2 , 3 , 4
   The Points for crossover between parameter 2 and 3 were:: 5 , 6 , 7
    [11] [10] [76]  Which equals   [00001011][00001010][01001100] in binary

   The Points for crossover between parameter 1 and 2 were :: 3 , 4 , 5
   The Points for crossover between parameter 2 and 3 were:: 6 , 7 , 8
    [9] [10] [9]  Which equals   [00001001][00001010][00001001] in binary

_____________________________________Crossover has finished :
```

Iteration 2 - Crossover

```
The Proposed Multi Mutation Operation :
_____________________________________

After Normal Single Mutation on the test data  [10]  [37]  [34] by flipping one gene randomly, The produced test data IS::
[42] [45] [38]

[42] [45] [38]
This Test Data covers non-duplicated path which is:: Path 4 , Rank= 0.2


After Normal Single Mutation on the test data  [0]  [79]  [11] by flipping one gene randomly, The produced test data IS::
[16] [15] [3]

[16] [15] [3]
This Test Data covers non-duplicated path which is:: Path 1 , Rank= 0.2


After Normal Single Mutation on the test data  [40]  [9]  [67] by flipping one gene randomly, The produced test data IS::
[104] [25] [83]

[104] [25] [83]
This Test Data covers non-duplicated path which is:: Path 2 , Rank= 0.2
```

Iteration 2 - The Proposed Repetitive Mutation -Part1-

```
After Normal Single Mutation on the test data  [11]  [10]  [76] by flipping one gene randomly, The produced test data IS::
[10] [14] [12]
This Test Data covers duplicated path Which is:: Path 4 , Rank= 0.166667

After Mutation ( 1 ) on the same test data by flipping one gene randomly, the new test data IS::
[3] [74] [78]

[3] [74] [78]
This Test Data covers non-duplicated path which is:: Path 5 , Rank= 0.2


After Normal Single Mutation on the test data  [9]  [10]  [9] by flipping one gene randomly, The produced test data IS::
[13] [2] [25]

[13] [2] [25]
This Test Data covers non-duplicated path which is:: Path 3 , Rank= 0.2


5 paths out of 5 paths have been covered,  Path coverage rate= 100%

 The number of test data iterations is 2
_____________________________________________________________

the fittest test data for path coverage testing are::

 [42] [45] [38] -----> The Covered Path is:: Path 4
_____________________________________________

 [16] [15] [3] -----> The Covered Path is:: Path 1
_____________________________________________

 [104] [25] [83] -----> The Covered Path is:: Path 2
_____________________________________________

 [3] [74] [78] -----> The Covered Path is:: Path 5
_____________________________________________

 [13] [2] [25] -----> The Covered Path is:: Path 3
_____________________________________________


Process returned 0 (0x0)   execution time : 2.204 s
Press any key to continue.
```

Iteration 2 - The Proposed Repetitive Mutation -Part2-

Appendix C

The Results' Snapshots of Single Loop SUT Experiment

```
Enter 1 For Sequential Experiment
Enter 2 For Single Loop Experiment
Enter 3 For Nested Loops Experiment
2
Enter CDC Value
5
```

The Simulation Program's Inputs

```
The Initial Selected Test Data are::

[8] [9] -----> The Covered Path is:: Path 1
_____________________________________

[5] [4] -----> The Covered Path is:: Path 3
_____________________________________

[2] [9] -----> The Covered Path is:: Path 1
_____________________________________

[4] [9] -----> The Covered Path is:: Path 1
_____________________________________

2 paths out of 4 paths have been covered and the coverage rate= 50%
```

The Initial Test Data


```
 Iteration 1
_____________


(The probability of crossover has not achieved)
```

Iteration 1 – Crossover's Probability has not Achieved

```
The Proposed Multi Mutation Operation :
_____________________________________

After Normal Single Mutation on the test data  [8]  [9] by flipping one gene randomly, The produced test data IS::
[9] [1]
[9] [1]
This Test Data covers non-duplicated path which is:: Path 4 , Rank= 0.25


After Normal Single Mutation on the test data  [5]  [4] by flipping one gene randomly, The produced test data IS::
[13] [68]
[13] [68]
This Test Data covers non-duplicated path which is:: Path 1 , Rank= 0.25


After Normal Single Mutation on the test data  [2]  [9] by flipping one gene randomly, The produced test data IS::
[34] [137]   This Test Data covers duplicated path Which is:: Path 1 , Rank= 0.2

After Mutation ( 1 ) on the same test data by flipping one gene randomly, the new test data IS::
[6] [73]   This Test Data covers duplicated path Which is:: Path 1 , Rank= 0.166667

After Mutation ( 2 ) on the same test data by flipping one gene randomly, the new test data IS::
[130] [13]   This Test Data covers duplicated path Which is:: Path 4 , Rank= 0.142857

After Mutation ( 3 ) on the same test data by flipping one gene randomly, the new test data IS::
[130] [73]
[130] [73]
This Test Data covers non-duplicated path which is:: Path 3 , Rank= 0.25
```

Iteration 1 - The Proposed Repetitive Mutation -Part1-

```
After Normal Single Mutation on the test data  [4]  [9] by flipping one gene randomly, The produced test data IS::
[68] [1]   This Test Data covers duplicated path Which is:: Path 4 , Rank= 0.2

After Mutation ( 1 ) on the same test data by flipping one gene randomly, the new test data IS::
[20] [1]   This Test Data covers duplicated path Which is:: Path 4 , Rank= 0.166667

After Mutation ( 2 ) on the same test data by flipping one gene randomly, the new test data IS::
[36] [41]   This Test Data covers duplicated path Which is:: Path 1 , Rank= 0.142857

After Mutation ( 3 ) on the same test data by flipping one gene randomly, the new test data IS::
[36] [41]   This Test Data covers duplicated path Which is:: Path 1 , Rank= 0.125

After Mutation ( 4 ) on the same test data by flipping one gene randomly, the new test data IS::
[36] [1]   This Test Data covers duplicated path Which is:: Path 4 , Rank= 0.111111

After Mutation ( 5 ) on the same test data by flipping one gene randomly, the new test data IS::
[36] [11]   This Test Data covers duplicated path Which is:: Path 3 , Rank= 0.0909091


3 paths out of 4 paths have been covered,  Path coverage rate= 75%

The number of test data iterations is 1
```

Iteration 1 - The Proposed Repetitive Mutation -Part2-

```
Iteration 2
_____________

The Selected Test Data are::

[9] [1]
[13] [68]
[130] [73]
[4] [9]
```

Iteration 2 - The Selected Test Data

```
The Crossover Operation :
_________________________

Test Data Before Crossover ::

    [9] [1]  Which equals   [00001001][00000001] in binary

    [13] [68]  Which equals   [00001101][01000100] in binary

    [130] [73]  Which equals   [10000010][01001001] in binary

    [4] [9]  Which equals   [00000100][00001001] in binary


Test Data After Crossover ::

   The Points for crossover were :: 1 , 2 , 3
    [9] [1]  Which equals   [00001001][00000001] in binary

   The Points for crossover were :: 5 , 6 , 7
    [5] [76]  Which equals   [00000101][01001100] in binary

   The Points for crossover were :: 3 , 4 , 5
    [138] [65]  Which equals   [10001010][01000001] in binary

   The Points for crossover were :: 5 , 6 , 7
    [8] [5]  Which equals   [00001000][00000101] in binary

_____________________________________________Crossover has finished :
```

Iteration 2 - Crossover

```
The Proposed Multi Mutation Operation :
_____________________________________

After Normal Single Mutation on the test data  [9]  [1] by flipping one gene randomly, The produced test data IS::
[41] [17]
[41] [17]
This Test Data covers non-duplicated path which is:: Path 3 , Rank= 0.25


After Normal Single Mutation on the test data  [5]  [76] by flipping one gene randomly, The produced test data IS::
[4] [72]
[4] [72]
This Test Data covers non-duplicated path which is:: Path 2 , Rank= 0.25


After Normal Single Mutation on the test data  [138]  [65] by flipping one gene randomly, The produced test data IS::
[170] [69]   This Test Data covers duplicated path Which is:: Path 3 , Rank= 0.2

After Mutation ( 1 ) on the same test data by flipping one gene randomly, the new test data IS::
[202] [193]   This Test Data covers duplicated path Which is:: Path 3 , Rank= 0.166667

After Mutation ( 2 ) on the same test data by flipping one gene randomly, the new test data IS::
[202] [64]   This Test Data covers duplicated path Which is:: Path 3 , Rank= 0.142857

After Mutation ( 3 ) on the same test data by flipping one gene randomly, the new test data IS::
[136] [81]   This Test Data covers duplicated path Which is:: Path 3 , Rank= 0.125

After Mutation ( 4 ) on the same test data by flipping one gene randomly, the new test data IS::
[154] [81]   This Test Data covers duplicated path Which is:: Path 3 , Rank= 0.111111

After Mutation ( 5 ) on the same test data by flipping one gene randomly, the new test data IS::
[202] [193]   This Test Data covers duplicated path Which is:: Path 3 , Rank= 0.0909091


After Normal Single Mutation on the test data  [8]  [5] by flipping one gene randomly, The produced test data IS::
[40] [1]
[40] [1]
This Test Data covers non-duplicated path which is:: Path 4 , Rank= 0.25


3 paths out of 4 paths have been covered,  Path coverage rate= 75%

 The number of test data iterations is 2
```

Iteration 2 - The Proposed Repetitive Mutation

```
Iteration 3
_____________

The Selected Test Data are::

[41] [17]
[4] [72]
[138] [65]
[40] [1]
```

Iteration 3 - The Selected Test Data

```
The Crossover Operation :
_________________________

Test Data Before Crossover ::

    [41] [17]  Which equals   [00101001][00010001] in binary

    [4] [72]  Which equals   [00000100][01001000] in binary

    [138] [65]  Which equals   [10001010][01000001] in binary

    [40] [1]  Which equals   [00101000][00000001] in binary


Test Data After Crossover ::

   The Points for crossover were :: 5 , 6 , 7
    [33] [25]  Which equals   [00100001][00011001] in binary

   The Points for crossover were :: 3 , 4 , 5
    [12] [64]  Which equals   [00001100][01000000] in binary

   The Points for crossover were :: 3 , 4 , 5
    [130] [73]  Which equals   [10000010][01001001] in binary

   The Points for crossover were :: 5 , 6 , 7
    [32] [9]  Which equals   [00100000][00001001] in binary

_____________________________________Crossover has finished :
```

Iteration 3 - Crossover

```
The Proposed Multi Mutation Operation :
_______________________________

After Normal Single Mutation on the test data  [33]  [25] by flipping one gene randomly, The produced test data IS::
[97] [153]
[97] [153]
This Test Data covers non-duplicated path which is:: Path 1 , Rank= 0.25


After Normal Single Mutation on the test data  [12]  [64] by flipping one gene randomly, The produced test data IS::
[140] [72]
[140] [72]
This Test Data covers non-duplicated path which is:: Path 3 , Rank= 0.25


After Normal Single Mutation on the test data  [130]  [73] by flipping one gene randomly, The produced test data IS::
[194] [65]   This Test Data covers duplicated path Which is:: Path 3 , Rank= 0.2

After Mutation ( 1 ) on the same test data by flipping one gene randomly, the new test data IS::
[131] [65]   This Test Data covers duplicated path Which is:: Path 3 , Rank= 0.166667

After Mutation ( 2 ) on the same test data by flipping one gene randomly, the new test data IS::
[194] [89]   This Test Data covers duplicated path Which is:: Path 3 , Rank= 0.142857

After Mutation ( 3 ) on the same test data by flipping one gene randomly, the new test data IS::
[131] [105]   This Test Data covers duplicated path Which is:: Path 3 , Rank= 0.125

After Mutation ( 4 ) on the same test data by flipping one gene randomly, the new test data IS::
[138] [105]   This Test Data covers duplicated path Which is:: Path 3 , Rank= 0.111111

After Mutation ( 5 ) on the same test data by flipping one gene randomly, the new test data IS::
[128] [75]   This Test Data covers duplicated path Which is:: Path 3 , Rank= 0.0909091
```

Iteration 3 - The Proposed Repetitive Mutation -Part1-

```
After Normal Single Mutation on the test data  [32]  [9] by flipping one gene randomly, The produced test data IS::
[34] [41]   This Test Data covers duplicated path Which is:: Path 1 , Rank= 0.2

After Mutation ( 1 ) on the same test data by flipping one gene randomly, the new test data IS::
[160] [25]   This Test Data covers duplicated path Which is:: Path 3 , Rank= 0.166667

After Mutation ( 2 ) on the same test data by flipping one gene randomly, the new test data IS::
[34] [73]   This Test Data covers duplicated path Which is:: Path 1 , Rank= 0.142857

After Mutation ( 3 ) on the same test data by flipping one gene randomly, the new test data IS::
[33] [8]   This Test Data covers duplicated path Which is:: Path 3 , Rank= 0.125

After Mutation ( 4 ) on the same test data by flipping one gene randomly, the new test data IS::
[33] [73]   This Test Data covers duplicated path Which is:: Path 1 , Rank= 0.111111

After Mutation ( 5 ) on the same test data by flipping one gene randomly, the new test data IS::
[34] [1]   This Test Data covers duplicated path Which is:: Path 4 , Rank= 0.0909091


3 paths out of 4 paths have been covered,  Path coverage rate= 75%

The number of test data iterations is 3
```

Iteration 3 - The Proposed Repetitive Mutation -Part2-

```
Iteration 4
_____________

The Selected Test Data are::

[97] [153]
[140] [72]
[130] [73]
[32] [9]
```

Iteration 4 - The Selected Test Data

```
(The probability of crossover has not achieved)
```

Iteration 4 – Crossover's Probability has not Achieved

```
The Proposed Multi Mutation Operation :
_____________________________________

After Normal Single Mutation on the test data  [97]  [153] by flipping one gene randomly, The produced test data IS::
[113] [155]
[113] [155]
This Test Data covers non-duplicated path which is:: Path 1 , Rank= 0.25


After Normal Single Mutation on the test data  [140]  [72] by flipping one gene randomly, The produced test data IS::
[156] [8]
[156] [8]
This Test Data covers non-duplicated path which is:: Path 3 , Rank= 0.25


After Normal Single Mutation on the test data  [130]  [73] by flipping one gene randomly, The produced test data IS::
[128] [9]   This Test Data covers duplicated path Which is:: Path 3 , Rank= 0.2

After Mutation ( 1 ) on the same test data by flipping one gene randomly, the new test data IS::
[194] [75]   This Test Data covers duplicated path Which is:: Path 3 , Rank= 0.166667

After Mutation ( 2 ) on the same test data by flipping one gene randomly, the new test data IS::
[138] [201]   This Test Data covers duplicated path Which is:: Path 1 , Rank= 0.142857

After Mutation ( 3 ) on the same test data by flipping one gene randomly, the new test data IS::
[134] [201]   This Test Data covers duplicated path Which is:: Path 1 , Rank= 0.125

After Mutation ( 4 ) on the same test data by flipping one gene randomly, the new test data IS::
[194] [65]   This Test Data covers duplicated path Which is:: Path 3 , Rank= 0.111111

After Mutation ( 5 ) on the same test data by flipping one gene randomly, the new test data IS::
[162] [89]   This Test Data covers duplicated path Which is:: Path 3 , Rank= 0.0909091
```

Iteration 4 - The Proposed Repetitive Mutation -Part1-

```
After Normal Single Mutation on the test data  [32]  [9] by flipping one gene randomly, The produced test data IS::
[160] [73]   This Test Data covers duplicated path Which is:: Path 3 , Rank= 0.2

After Mutation ( 1 ) on the same test data by flipping one gene randomly, the new test data IS::
[33] [11]
[33] [11]
This Test Data covers non-duplicated path which is:: Path 4 , Rank= 0.25


3 paths out of 4 paths have been covered,  Path coverage rate= 75%

The number of test data iterations is 4
```

Iteration 4 - The Proposed Repetitive Mutation -Part2-

```
Iteration 5
_____________

The Selected Test Data are::

[113] [155]
[156] [8]
[130] [73]
[33] [11]
```

Iteration 5 - The Selected Test Data

```
The Crossover Operation :
_________________________

Test Data Before Crossover ::

    [113] [155]  Which equals   [01110001][10011011] in binary

    [156] [8]  Which equals   [10011100][00001000] in binary

    [130] [73]  Which equals   [10000010][01001001] in binary

    [33] [11]  Which equals   [00100001][00001011] in binary


Test Data After Crossover ::

    The Points for crossover were :: 6 , 7 , 8
     [115] [153]  Which equals   [01110011][10011001] in binary

    The Points for crossover were :: 4 , 5 , 6
     [136] [28]  Which equals   [10001000][00011100] in binary

    The Points for crossover were :: 5 , 6 , 7
     [136] [67]  Which equals   [10001000][01000011] in binary

    The Points for crossover were :: 2 , 3 , 4
     [1] [43]  Which equals   [00000001][00101011] in binary

_____________________________________Crossover has finished :
```

Iteration 5 - Crossover

```
The Proposed Multi Mutation Operation :
_____________________________________

After Normal Single Mutation on the test data  [115]  [153] by flipping one gene randomly, The produced test data IS::
[114] [185]
[114] [185]
This Test Data covers non-duplicated path which is:: Path 1 , Rank= 0.25


After Normal Single Mutation on the test data  [136]  [28] by flipping one gene randomly, The produced test data IS::
[140] [156]   This Test Data covers duplicated path Which is:: Path 1 , Rank= 0.2

After Mutation ( 1 ) on the same test data by flipping one gene randomly, the new test data IS::
[138] [12]
[138] [12]
This Test Data covers non-duplicated path which is:: Path 3 , Rank= 0.25


After Normal Single Mutation on the test data  [136]  [67] by flipping one gene randomly, The produced test data IS::
[168] [75]   This Test Data covers duplicated path Which is:: Path 3 , Rank= 0.2

After Mutation ( 1 ) on the same test data by flipping one gene randomly, the new test data IS::
[137] [99]   This Test Data covers duplicated path Which is:: Path 3 , Rank= 0.166667

After Mutation ( 2 ) on the same test data by flipping one gene randomly, the new test data IS::
[152] [65]   This Test Data covers duplicated path Which is:: Path 3 , Rank= 0.142857

After Mutation ( 3 ) on the same test data by flipping one gene randomly, the new test data IS::
[200] [83]   This Test Data covers duplicated path Which is:: Path 3 , Rank= 0.125

After Mutation ( 4 ) on the same test data by flipping one gene randomly, the new test data IS::
[152] [65]   This Test Data covers duplicated path Which is:: Path 3 , Rank= 0.111111

After Mutation ( 5 ) on the same test data by flipping one gene randomly, the new test data IS::
[138] [195]   This Test Data covers duplicated path Which is:: Path 1 , Rank= 0.0909091
```

Iteration 5 - The Proposed Repetitive Mutation -Part1-

```
After Normal Single Mutation on the test data  [1]  [43] by flipping one gene randomly, The produced test data IS::
[9] [59]   This Test Data covers duplicated path Which is:: Path 1 , Rank= 0.2

After Mutation ( 1 ) on the same test data by flipping one gene randomly, the new test data IS::
[65] [59]   This Test Data covers duplicated path Which is:: Path 3 , Rank= 0.166667

After Mutation ( 2 ) on the same test data by flipping one gene randomly, the new test data IS::
[129] [47]   This Test Data covers duplicated path Which is:: Path 3 , Rank= 0.142857

After Mutation ( 3 ) on the same test data by flipping one gene randomly, the new test data IS::
[3] [47]   This Test Data covers duplicated path Which is:: Path 1 , Rank= 0.125

After Mutation ( 4 ) on the same test data by flipping one gene randomly, the new test data IS::
[17] [47]   This Test Data covers duplicated path Which is:: Path 1 , Rank= 0.111111

After Mutation ( 5 ) on the same test data by flipping one gene randomly, the new test data IS::
[5] [59]   This Test Data covers duplicated path Which is:: Path 1 , Rank= 0.0909091


2 paths out of 4 paths have been covered,  Path coverage rate= 50%

The number of test data iterations is 5
```

Iteration 5 - The Proposed Repetitive Mutation -Part2-

```
Iteration 6
_____________

The Selected Test Data are::

[113] [155]
[156] [8]
[130] [73]
[33] [11]
```

Iteration 6 - The Selected Test Data

```
The Crossover Operation :
_________________________

Test Data Before Crossover ::

    [113] [155]  Which equals   [01110001][10011011] in binary

    [156] [8]  Which equals   [10011100][00001000] in binary

    [130] [73]  Which equals   [10000010][01001001] in binary

    [33] [11]  Which equals   [00100001][00001011] in binary

Test Data After Crossover ::

   The Points for crossover were :: 3 , 4 , 5
    [89] [179]  Which equals   [01011001][10110011] in binary

   The Points for crossover were :: 4 , 5 , 6
    [136] [28]  Which equals   [10001000][00011100] in binary

   The Points for crossover were :: 2 , 3 , 4
    [194] [9]  Which equals   [11000010][00001001] in binary

   The Points for crossover were :: 4 , 5 , 6
    [41] [3]  Which equals   [00101001][00000011] in binary

_____________________________________Crossover has finished :
```

Iteration 6 - Crossover

```
(The probability of mutation has not achieved)

 [89] [179]  -----> The Covered Path is:: Path 1
_____________________________________________

 [136] [28]  -----> The Covered Path is:: Path 3
_____________________________________________

 [194] [9]  -----> The Covered Path is:: Path 3
_____________________________________________

 [41] [3]  -----> The Covered Path is:: Path 3
_____________________________________________

2 paths out of 4 paths have been covered,  Path coverage rate= 50%
 The number of test data iterations is 6
```

Iteration 6 – The Proposed Repetitive Mutation's Probability has not Achieved

```
Iteration 7
_____________

The Selected Test Data are::

[113] [155]
[156] [8]
[130] [73]
[33] [11]
```

Iteration 7 - The Selected Test Data

```
The Crossover Operation :
_________________________

Test Data Before Crossover ::

    [113] [155]  Which equals   [01110001][10011011] in binary

    [156] [8]  Which equals   [10011100][00001000] in binary

    [130] [73]  Which equals   [10000010][01001001] in binary

    [33] [11]  Which equals   [00100001][00001011] in binary


Test Data After Crossover ::

   The Points for crossover were :: 6 , 7 , 8
    [115] [153]  Which equals   [01110011][10011001] in binary

   The Points for crossover were :: 2 , 3 , 4
    [140] [24]  Which equals   [10001100][00011000] in binary

   The Points for crossover were :: 5 , 6 , 7
    [136] [67]  Which equals   [10001000][01000011] in binary

   The Points for crossover were :: 5 , 6 , 7
    [43] [1]  Which equals   [00101011][00000001] in binary

_____________________________________________Crossover has finished :
```

Iteration 7 - Crossover

```
(The probability of mutation has not achieved)

 [115] [153]  -----> The Covered Path is:: Path 1
_____________________________________________

 [140] [24]  -----> The Covered Path is:: Path 3
_____________________________________________

 [136] [67]  -----> The Covered Path is:: Path 3
_____________________________________________

 [43] [1]  -----> The Covered Path is:: Path 4
_____________________________________________

3 paths out of 4 paths have been covered,  Path coverage rate= 75%
 The number of test data iterations is 7
```

Iteration 7 – The Proposed Repetitive Mutation's Probability has not Achieved

```
Iteration 8
_____________

The Selected Test Data are::

[115] [153]
[140] [24]
[136] [67]
[43] [1]
```

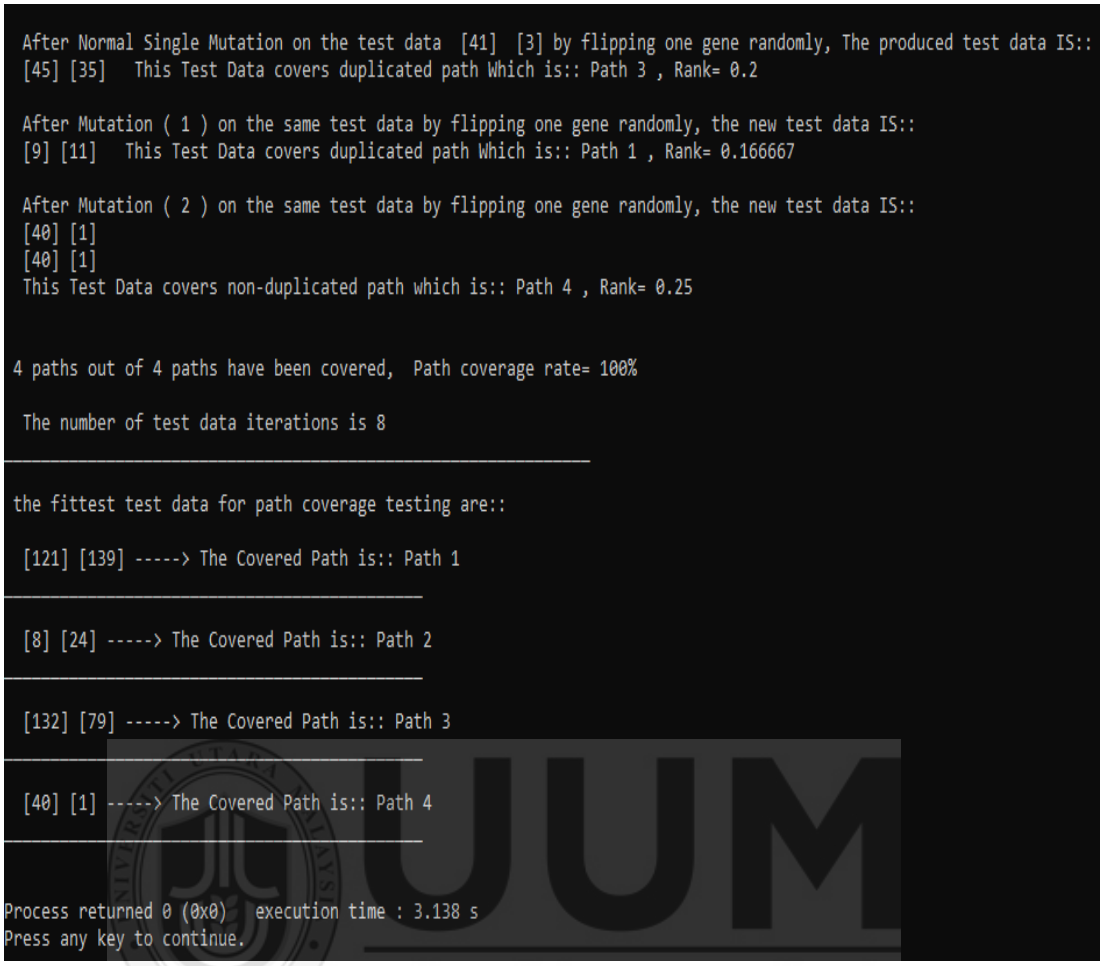
Iteration 8 - The Selected Test Data

```
The Crossover Operation :
_________________________

Test Data Before Crossover ::

    [115] [153]  Which equals   [01110011][10011001] in binary

    [140] [24]  Which equals   [10001100][00011000] in binary

    [136] [67]  Which equals   [10001000][01000011] in binary

    [43] [1]  Which equals   [00101011][00000001] in binary


Test Data After Crossover ::

   The Points for crossover were :: 6 , 7 , 8
    [113] [155]  Which equals   [01110001][10011011] in binary

   The Points for crossover were :: 1 , 2 , 3
    [12] [152]  Which equals   [00001100][10011000] in binary

   The Points for crossover were :: 4 , 5 , 6
    [128] [75]  Which equals   [10000000][01001011] in binary

   The Points for crossover were :: 6 , 7 , 8
    [41] [3]  Which equals   [00101001][00000011] in binary

                                        _____
_____________________________________Crossover has finished :
```


Iteration 8 - Crossover

```
The Proposed Multi Mutation Operation :
_______________________________________

After Normal Single Mutation on the test data  [113]  [155] by flipping one gene randomly, The produced test data IS::
[121] [139]
[121] [139]
This Test Data covers non-duplicated path which is:: Path 1 , Rank= 0.25


After Normal Single Mutation on the test data  [12]  [152] by flipping one gene randomly, The produced test data IS::
[8] [24]
[8] [24]
This Test Data covers non-duplicated path which is:: Path 2 , Rank= 0.25


After Normal Single Mutation on the test data  [128]  [75] by flipping one gene randomly, The produced test data IS::
[132] [79]
[132] [79]
This Test Data covers non-duplicated path which is:: Path 3 , Rank= 0.25
```

Iteration 8 - The Proposed Repetitive Mutation -Part1-



Iteration 8 - The Proposed Repetitive Mutation -Part2-

Appendix D

The Results' Snapshots of Nested Loops SUT Experiment

```
Enter 1 For Sequential Experiment
Enter 2 For Single Loop Experiment
Enter 3 For Nested Loops Experiment
3
Enter CDC Value
10
```

The Simulation Program's Inputs

```
The Initial Selected Test Data are::

[6] [1] -----> The Covered Path is:: Path 4
_____________________________________________

[8] [3] -----> The Covered Path is:: Path 4
_____________________________________________

[4] [3] -----> The Covered Path is:: Path 5
_____________________________________________

[7] [2] -----> The Covered Path is:: Path 4
_____________________________________________

[9] [9] -----> The Covered Path is:: Path 5
_____________________________________________

2 paths out of 5 paths have been covered and the coverage rate= 40%
```

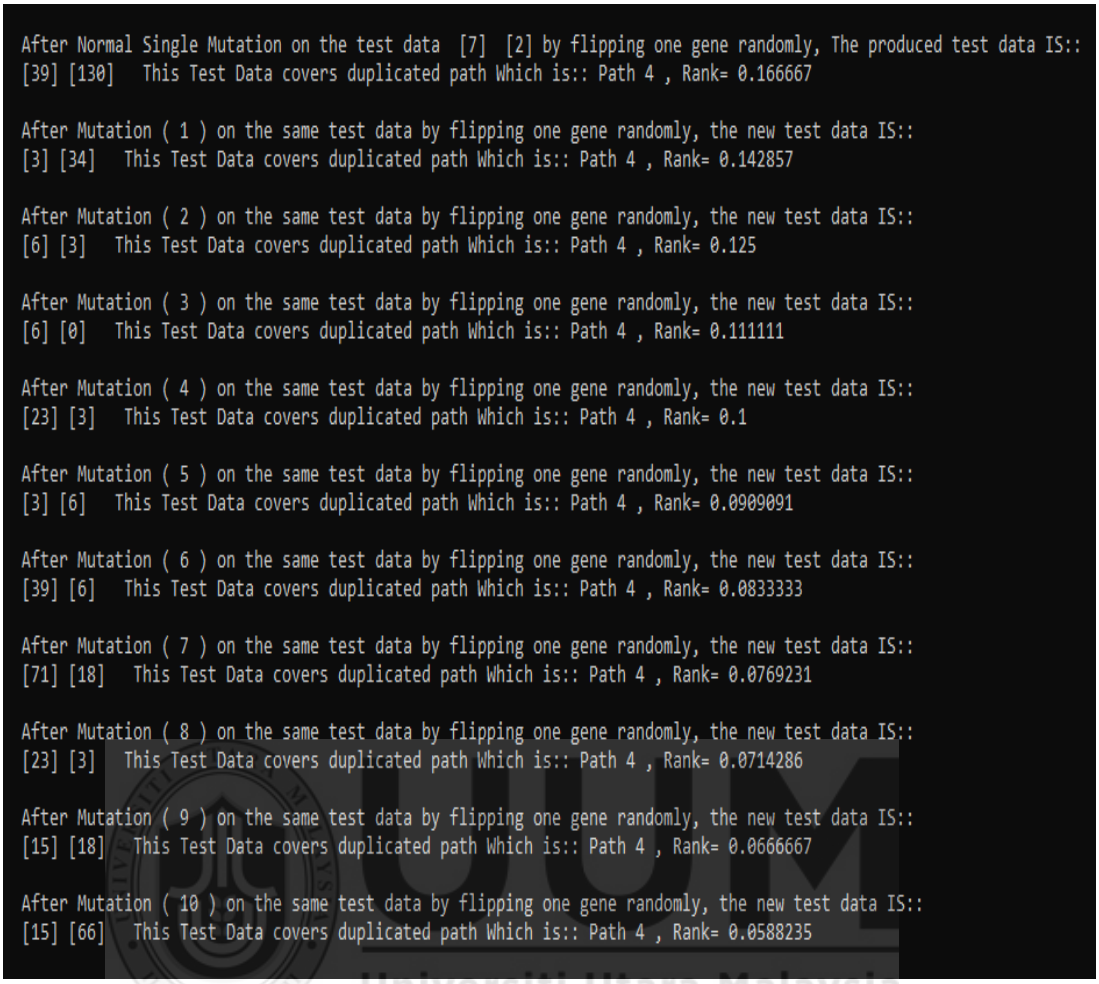
The Initial Test Data

```
Iteration 1
_____________


The Crossover Operation :
_________________________

Test Data Before Crossover ::

    [6] [1]  Which equals   [00000110][00000001] in binary

    [8] [3]  Which equals   [00001000][00000011] in binary

    [4] [3]  Which equals   [00000100][00000011] in binary

    [7] [2]  Which equals   [00000111][00000010] in binary

    [9] [9]  Which equals   [00001001][00001001] in binary


Test Data After Crossover ::

   The Points for crossover were :: 2 , 3 , 4
    [6] [1]  Which equals   [00000110][00000001] in binary

   The Points for crossover were :: 6 , 7 , 8
    [11] [0]  Which equals   [00001011][00000000] in binary

   The Points for crossover were :: 4 , 5 , 6
    [0] [7]  Which equals   [00000000][00000111] in binary

   The Points for crossover were :: 3 , 4 , 5
    [7] [2]  Which equals   [00000111][00000010] in binary

   The Points for crossover were :: 1 , 2 , 3
    [9] [9]  Which equals   [00001001][00001001] in binary

_____________________________________Crossover has finished :
```

Iteration 1 - Crossover

```
The Proposed Multi Mutation Operation :
_______________________________________

After Normal Single Mutation on the test data  [6]  [1] by flipping one gene randomly, The produced test data IS::
[38] [9]
[38] [9]
This Test Data covers non-duplicated path which is:: Path 4 , Rank= 0.2


After Normal Single Mutation on the test data  [11]  [0] by flipping one gene randomly, The produced test data IS::
[10] [8]
[10] [8]
This Test Data covers non-duplicated path which is:: Path 3 , Rank= 0.2


After Normal Single Mutation on the test data  [0]  [7] by flipping one gene randomly, The produced test data IS::
[16] [39]   This Test Data covers duplicated path Which is:: Path 4 , Rank= 0.166667

After Mutation ( 1 ) on the same test data by flipping one gene randomly, the new test data IS::
[16] [15]
[16] [15]
This Test Data covers non-duplicated path which is:: Path 5 , Rank= 0.2
```

Iteration 1 - The Proposed Repetitive Mutation -Part1-



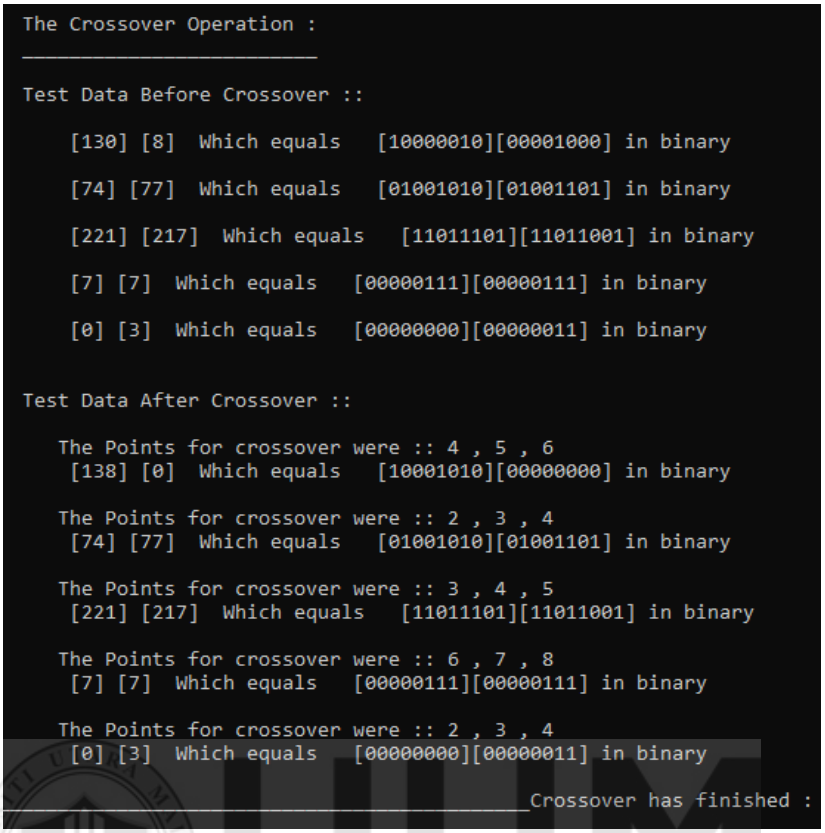
Iteration 1 - The Proposed Repetitive Mutation -Part2-

```
After Normal Single Mutation on the test data  [9]  [9] by flipping one gene randomly, The produced test data IS::
[137] [137]   This Test Data covers duplicated path Which is:: Path 5 , Rank= 0.166667

After Mutation ( 1 ) on the same test data by flipping one gene randomly, the new test data IS::
[137] [1]   This Test Data covers duplicated path Which is:: Path 4 , Rank= 0.142857

After Mutation ( 2 ) on the same test data by flipping one gene randomly, the new test data IS::
[11] [25]   This Test Data covers duplicated path Which is:: Path 4 , Rank= 0.125

After Mutation ( 3 ) on the same test data by flipping one gene randomly, the new test data IS::
[11] [1]   This Test Data covers duplicated path Which is:: Path 4 , Rank= 0.111111

After Mutation ( 4 ) on the same test data by flipping one gene randomly, the new test data IS::
[41] [1]   This Test Data covers duplicated path Which is:: Path 4 , Rank= 0.1

After Mutation ( 5 ) on the same test data by flipping one gene randomly, the new test data IS::
[11] [25]   This Test Data covers duplicated path Which is:: Path 4 , Rank= 0.0909091

After Mutation ( 6 ) on the same test data by flipping one gene randomly, the new test data IS::
[137] [137]   This Test Data covers duplicated path Which is:: Path 5 , Rank= 0.0833333

After Mutation ( 7 ) on the same test data by flipping one gene randomly, the new test data IS::
[41] [11]   This Test Data covers duplicated path Which is:: Path 4 , Rank= 0.0769231

After Mutation ( 8 ) on the same test data by flipping one gene randomly, the new test data IS::
[11] [13]   This Test Data covers duplicated path Which is:: Path 3 , Rank= 0.0714286

After Mutation ( 9 ) on the same test data by flipping one gene randomly, the new test data IS::
[41] [1]   This Test Data covers duplicated path Which is:: Path 4 , Rank= 0.0666667

After Mutation ( 10 ) on the same test data by flipping one gene randomly, the new test data IS::
[137] [8]   This Test Data covers duplicated path Which is:: Path 4 , Rank= 0.0588235


3 paths out of 5 paths have been covered,  Path coverage rate= 60%

The number of test data iterations is 1
```


Iteration 1 - The Proposed Repetitive Mutation -Part3-

```
Iteration 25
_____________

The Selected Test Data are::

[130] [8]
[74] [77]
[221] [217]
[7] [7]
[0] [3]
```

Iteration 25 - The Selected Test Data



Iteration 25 - Crossover

Iteration 25 – The Proposed Repetitive Mutation's Probability has not Achieved

```
Iteration 26
_____________

The Selected Test Data are::

[138] [0]
[74] [77]
[221] [217]
[7] [7]
[0] [3]
```

Iteration 26 - The Selected Test Data

```
The Crossover Operation :
_________________________

Test Data Before Crossover ::

    [138] [0]  Which equals   [10001010][00000000] in binary

    [74] [77]  Which equals   [01001010][01001101] in binary

    [221] [217]  Which equals   [11011101][11011001] in binary

    [7] [7]  Which equals   [00000111][00000111] in binary

    [0] [3]  Which equals   [00000000][00000011] in binary


Test Data After Crossover ::

   The Points for crossover were :: 6 , 7 , 8
    [136] [2]  Which equals   [10001000][00000010] in binary

   The Points for crossover were :: 5 , 6 , 7
    [76] [75]  Which equals   [01001100][01001011] in binary

   The Points for crossover were :: 6 , 7 , 8
    [217] [221]  Which equals   [11011001][11011101] in binary

   The Points for crossover were :: 5 , 6 , 7
    [7] [7]  Which equals   [00000111][00000111] in binary

   The Points for crossover were :: 4 , 5 , 6
    [0] [3]  Which equals   [00000000][00000011] in binary

_____________________________________Crossover has finished :
```

Iteration 26 - Crossover

```
The Proposed Multi Mutation Operation :
_____________________________________

After Normal Single Mutation on the test data  [136]  [2] by flipping one gene randomly, The produced test data IS::
[140] [66]
[140] [66]
This Test Data covers non-duplicated path which is:: Path 4 , Rank= 0.2


After Normal Single Mutation on the test data  [76]  [75] by flipping one gene randomly, The produced test data IS::
[72] [11]   This Test Data covers duplicated path Which is:: Path 4 , Rank= 0.166667

After Mutation ( 1 ) on the same test data by flipping one gene randomly, the new test data IS::
[92] [91]
[92] [91]
This Test Data covers non-duplicated path which is:: Path 5 , Rank= 0.2


After Normal Single Mutation on the test data  [217]  [221] by flipping one gene randomly, The produced test data IS::
[201] [93]   This Test Data covers duplicated path Which is:: Path 4 , Rank= 0.166667

After Mutation ( 1 ) on the same test data by flipping one gene randomly, the new test data IS::
[209] [157]   This Test Data covers duplicated path Which is:: Path 4 , Rank= 0.142857

After Mutation ( 2 ) on the same test data by flipping one gene randomly, the new test data IS::
[209] [205]
[209] [205]
This Test Data covers non-duplicated path which is:: Path 3 , Rank= 0.2
```

Iteration 26 - The Proposed Repetitive Mutation -Part1-

```
After Normal Single Mutation on the test data  [7]  [7] by flipping one gene randomly, The produced test data IS::
[135] [6]   This Test Data covers duplicated path Which is:: Path 4 , Rank= 0.166667

After Mutation ( 1 ) on the same test data by flipping one gene randomly, the new test data IS::
[135] [15]   This Test Data covers duplicated path Which is:: Path 4 , Rank= 0.142857

After Mutation ( 2 ) on the same test data by flipping one gene randomly, the new test data IS::
[71] [3]   This Test Data covers duplicated path Which is:: Path 4 , Rank= 0.125

After Mutation ( 3 ) on the same test data by flipping one gene randomly, the new test data IS::
[71] [15]   This Test Data covers duplicated path Which is:: Path 4 , Rank= 0.111111

After Mutation ( 4 ) on the same test data by flipping one gene randomly, the new test data IS::
[6] [5]   This Test Data covers duplicated path Which is:: Path 5 , Rank= 0.1

After Mutation ( 5 ) on the same test data by flipping one gene randomly, the new test data IS::
[15] [3]   This Test Data covers duplicated path Which is:: Path 4 , Rank= 0.0909091

After Mutation ( 6 ) on the same test data by flipping one gene randomly, the new test data IS::
[71] [5]   This Test Data covers duplicated path Which is:: Path 4 , Rank= 0.0833333

After Mutation ( 7 ) on the same test data by flipping one gene randomly, the new test data IS::
[39] [71]   This Test Data covers duplicated path Which is:: Path 4 , Rank= 0.0769231

After Mutation ( 8 ) on the same test data by flipping one gene randomly, the new test data IS::
[135] [6]   This Test Data covers duplicated path Which is:: Path 4 , Rank= 0.0714286

After Mutation ( 9 ) on the same test data by flipping one gene randomly, the new test data IS::
[135] [39]   This Test Data covers duplicated path Which is:: Path 4 , Rank= 0.0666667

After Mutation ( 10 ) on the same test data by flipping one gene randomly, the new test data IS::
[5] [3]   This Test Data covers duplicated path Which is:: Path 3 , Rank= 0.0588235
```

Iteration 26 - The Proposed Repetitive Mutation -Part2-

```
After Normal Single Mutation on the test data  [0]  [3] by flipping one gene randomly, The produced test data IS::
[4] [67]   This Test Data covers duplicated path Which is:: Path 4 , Rank= 0.166667

After Mutation ( 1 ) on the same test data by flipping one gene randomly, the new test data IS::
[8] [35]   This Test Data covers duplicated path Which is:: Path 4 , Rank= 0.142857

After Mutation ( 2 ) on the same test data by flipping one gene randomly, the new test data IS::
[1] [1]   This Test Data covers duplicated path Which is:: Path 5 , Rank= 0.125

After Mutation ( 3 ) on the same test data by flipping one gene randomly, the new test data IS::
[8] [19]   This Test Data covers duplicated path Which is:: Path 4 , Rank= 0.111111

After Mutation ( 4 ) on the same test data by flipping one gene randomly, the new test data IS::
[16] [11]   This Test Data covers duplicated path Which is:: Path 4 , Rank= 0.1

After Mutation ( 5 ) on the same test data by flipping one gene randomly, the new test data IS::
[2] [35]   This Test Data covers duplicated path Which is:: Path 4 , Rank= 0.0909091

After Mutation ( 6 ) on the same test data by flipping one gene randomly, the new test data IS::
[2] [11]   This Test Data covers duplicated path Which is:: Path 4 , Rank= 0.0833333

After Mutation ( 7 ) on the same test data by flipping one gene randomly, the new test data IS::
[128] [67]   This Test Data covers duplicated path Which is:: Path 4 , Rank= 0.0769231

After Mutation ( 8 ) on the same test data by flipping one gene randomly, the new test data IS::
[32] [2]   This Test Data covers duplicated path Which is:: Path 4 , Rank= 0.0714286

After Mutation ( 9 ) on the same test data by flipping one gene randomly, the new test data IS::
[8] [2]   This Test Data covers duplicated path Which is:: Path 4 , Rank= 0.0666667

After Mutation ( 10 ) on the same test data by flipping one gene randomly, the new test data IS::
[1] [11]   This Test Data covers duplicated path Which is:: Path 4 , Rank= 0.0588235


3 paths out of 5 paths have been covered,  Path coverage rate= 60%

The number of test data iterations is 26
```


Iteration 26 - The Proposed Repetitive Mutation -Part3-

```
Iteration 27
_____________

The Selected Test Data are::

[138] [0]
[74] [77]
[221] [217]
[7] [7]
[0] [3]
```

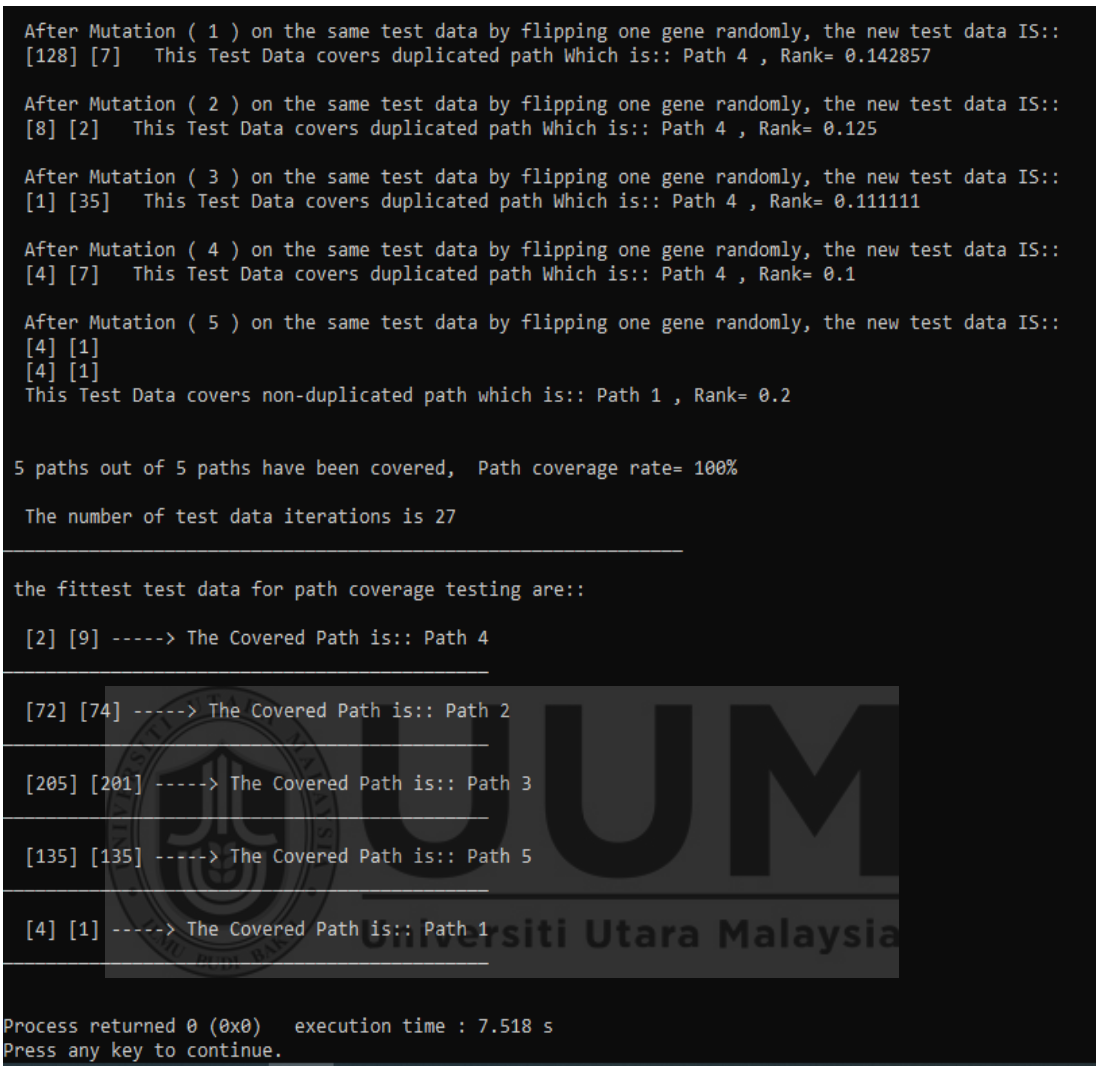
Iteration 27 - The Selected Test Data

```
The Crossover Operation :
_________________________

Test Data Before Crossover ::

    [138] [0]  Which equals   [10001010][00000000] in binary

    [74] [77]  Which equals   [01001010][01001101] in binary

    [221] [217]  Which equals   [11011101][11011001] in binary

    [7] [7]  Which equals   [00000111][00000111] in binary

    [0] [3]  Which equals   [00000000][00000011] in binary


Test Data After Crossover ::

   The Points for crossover were :: 4 , 5 , 6
    [130] [8]  Which equals   [10000010][00001000] in binary

   The Points for crossover were :: 5 , 6 , 7
    [76] [75]  Which equals   [01001100][01001011] in binary

   The Points for crossover were :: 1 , 2 , 3
    [221] [217]  Which equals   [11011101][11011001] in binary

   The Points for crossover were :: 3 , 4 , 5
    [7] [7]  Which equals   [00000111][00000111] in binary

   The Points for crossover were :: 2 , 3 , 4
    [0] [3]  Which equals   [00000000][00000011] in binary

_____________________________________Crossover has finished :
```

Iteration 27 - Crossover

```
The Proposed Multi Mutation Operation :
_______________________________

After Normal Single Mutation on the test data  [130]  [8] by flipping one gene randomly, The produced test data IS::
[2] [9]
[2] [9]
This Test Data covers non-duplicated path which is:: Path 4 , Rank= 0.2


After Normal Single Mutation on the test data  [76]  [75] by flipping one gene randomly, The produced test data IS::
[72] [74]
[72] [74]
This Test Data covers non-duplicated path which is:: Path 2 , Rank= 0.2


After Normal Single Mutation on the test data  [221]  [217] by flipping one gene randomly, The produced test data IS::
[157] [216]   This Test Data covers duplicated path Which is:: Path 4 , Rank= 0.166667

After Mutation ( 1 ) on the same test data by flipping one gene randomly, the new test data IS::
[205] [201]
[205] [201]
This Test Data covers non-duplicated path which is:: Path 3 , Rank= 0.2


After Normal Single Mutation on the test data  [7]  [7] by flipping one gene randomly, The produced test data IS::
[135] [135]
[135] [135]
This Test Data covers non-duplicated path which is:: Path 5 , Rank= 0.2


After Normal Single Mutation on the test data  [0]  [3] by flipping one gene randomly, The produced test data IS::
[8] [131]   This Test Data covers duplicated path Which is:: Path 4 , Rank= 0.166667
```

Iteration 27 - The Proposed Repetitive Mutation -Part1-



Iteration 27 - The Proposed Repetitive Mutation -Part2-