International Journal on Advanced Science Engineering Information Technology

Vol.12 (2022) No. 6
ISSN: 2088-5334

Optimization of Mutation Testing Challenges to Fixing Faults

Sasa Ani Arnomo [a,*], Noraini Binti Ibrahim [b], Anggia Dasa Putri [c], Ellbert Hutabri [c]

[a] *Department of Information System, Universitas Putera Batam, Batam, Indonesia*
[b] *Faculty Sciences Computer and Technology Information, Universiti Tun Hussein Onn Malaysia (UTHM), Johor, Malaysia*
[c] *Department of Informatics Engineering, Universitas Putera Batam, Batam, Indonesia*
*Corresponding author: *sasa@puterabatam.ac.id*

***Abstract*— One of the challenges of mutation testing is fixing faults. In the debugging phase, all live mutants were repaired. Programs need high mutation scores to be declared reliable program codes. Each mutation test can allow the identification of multiple mutants. This is what confuses the faults fixing process. The objective of this research is to get the shortest route so that it can help in sorting the mutant types during application improvement after testing. The optimization is needed considering the number of mutants in each mutation testing. The problems related to optimization are very complex. It takes a suitable method to find the shortest path by paying attention to each point. There are 30 projects chosen randomly. The operator mutations that are often killed when testing mutations are AOIU and COI. The proposed optimization for mutant repair sequence is the ant colony system (ACS). The route selection using the Ant Colony System algorithm resulted in route optimization of 1.528254. Meanwhile, if the genetic algorithm is used, the score is 1.767643. Optimization results are very helpful for developers in improving code in mutation testing. Research states the best order for handling mutants using ACS. This research can be further developed with the addition of class-level mutant cases which are produced using class mutation operators. Class mutation operators have different characteristics from traditional mutation operators. In particular, it requires changes to the program structure, such as the definition of class variables.**

***Keywords*— Fixing faults; mutation testing; optimization; ACS.**

Manuscript received 19 Feb. 2021; revised 31 Jul. 2021; accepted 8 Sep. 2021. Date of publication 31 Dec. 2022.

I. INTRODUCTION

Software testing is essential in creating high-quality software [1]–[3]. There are many types of testing software which include mutation testing. Mutation testing is a white box software testing technique based on faults [4], [5], which is used to assess the quality of a program's code. A number of mutants will be generated when testing a source code. Each mutant appears based on the manipulation of the original program through a transformed mutation operator [6], [7]. Mutation testing executes the mutants from the imitation program and determines whether the mutants can be killed. The test case generates a different set of tests that can be run to detect faults [8]. The results of the execution of an imitation program get a different value from the results of the original program execution, so it can be stated that the mutant status is killed [9], [10]. If the execution value is the same, the mutant is declared alive. Live mutants need to be evaluated where the original program with the imitation program has the same execution value even though they both have different codes. After testing, the mutation score can be calculated based on the number of live mutants and dead mutants. The mutation score is used in the research discussion.

Mutation testing has problems with computational costs[11]. This makes it possible to generate a large number of mutants upon execution on a test suite. The cost of creating the mutants and repairing the program is expensive. Researchers have proposed many techniques to reduce their costs[12], such as weak mutation testing [13], selective mutation testing, high mutation testing, mutant relationship redundancy[14], Model-based testing (MBT)[15], classification, clustering, and the advantage of high fault localization accuracy[16]. However, the cost of the mutation process remains high. This study proposes how to optimize the repair sequence of the mutants that have been detected. The data was recapitulated based on the mutation score for each mutant.

The mutant selection process is needed to measure the representation of the important selected mutants. In selective mutation testing, the selected mutants should represent all mutants that appear in the test series[17]. It gave effectiveness in testing capabilities that reveal the error code. Selecting mutants to subsets that inspire new test case designs is helpful in mutation testing. Several algorithms were developed, such as Evolutionary Mutation Testing (EMT) [18]. In general, there are three categories of mutant selection techniques: random-based mutant selection, operator-based mutant selection, and element-based mutant selection.

Some researchers optimize the creation of test cases to reduce costs using genetic algorithms[19]–[21]. Meanwhile, this algorithm still needs to be studied further, whether it has been presented in all mutants. An important step after mutation testing is code correction. Thus, this research tries to optimize the order of how mutant repair. Generally, mutation testing takes a lot of time for the programmer. Many test cases can be applied. Thus, testing gave rise to many mutants. This requires extra handlers. The selection of mutants is one thing that greatly affects the testing time and costs of mutation testing. The optimization technique used in this case is the Ant Colony System (ACS) algorithm. ACS found that the route cost and time are less than other optimization methods[22]. Ant Colony System (ACS) is an algorithm based on the route of the ants. In the ACS algorithm, the process of forming an ant travel path is applied to find a solution to the optimization problem. As a comparison of optimization, a genetic algorithm is chosen. A genetic algorithm is a solution-seeking technique that follows the natural selection of biological evolution [23].

II. Material and Method

A. Mutation Operator

The mutation operator is the rule for the changes that produce mutants [24]. This change is intended to prove the reliability of the program code. The mutation operator demands the adjustment of the programming language written on the program under test. This change can be the deletion, insertion, or replacement of an operator from the program statement. After the mutant is created, the original test suite will execute all its test cases on the modified version of the project [25]. Table 1 shows several types of mutation operators that can be used for testing.

TABLE I
MUTATION OPERATOR TABLE

Code	Mutation Operator
AOIS	Arithmetic Operator Insertion Short-cut
AOIU	Arithmetic Operator Insertion Unary
AORB	Arithmetic Operator Mutation Operator Description
AORS	Arithmetic Operator Replacement Short-Cut
ASRS	Short-Cut Assignment Operator Replacement
COD	Conditional Operator Deletion
COI	Conditional Operator Insertion
COR	Conditional Operator Replacement
IOD	Overriding Method Deletion
JID	Member Variable Initialization Deletion
JSI	Static Modifier Insertion
JTD	This Keyword Deletion
JTI	This Keyword Insertion
LOI	Logical Operator Insertion
OAN	Argument Number Changed

B. Mutation score

Mutation score can indicate a low or high value. The developer works hard to improve the program when the mutation score is low. Where the test found many errors by marking the number of mutants alive. A high mutation score means that the program has a good test suite [22] and a good code structure. The mutation score (MS) using the calculation formula is as follows [6] [23]:

$$MS = 100 * D / N \tag{1}$$

Where N is the total of mutants; D is the number of killed mutants. The high mutation score value is the mutation score getting closer to 1. The test data set shows that most of the mutants were killed.

C. Mutation testing challenge

Mutation testing has several research approaches. The approach is a mutant generation, test generation and execution, and Evaluation [20]. Figure 1 shows the mutation testing challenge. The emphasis of the study is the optimization of fixing the program through mutant sequences were found.

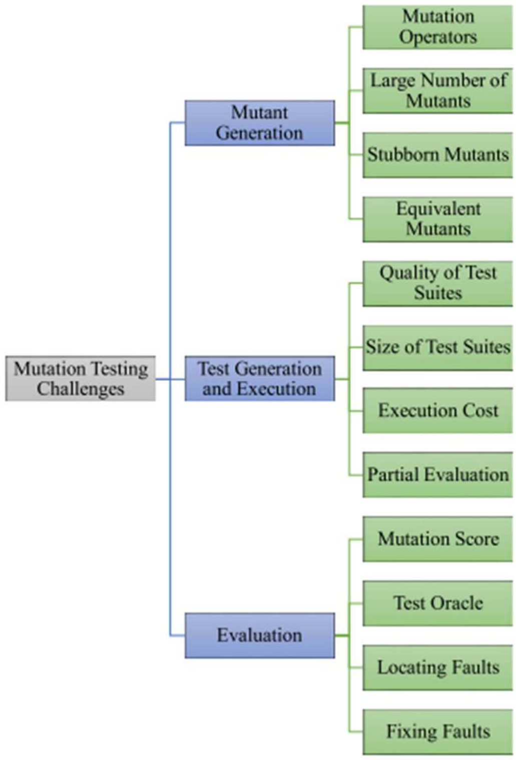

Fig. 1 Mutation Testing Challenge

D. How the Ant Algorithm Works to Find the Optimal Path.

Ants can sense their complex environment in search of food. Then the ants return to the nest through the path at the mark of the pheromone substance left behind. Pheromones are chemical substances that come from endocrine glands. The process of pheromone inheritance is known as stigmergy. It is marking an area to create a route to the nest. Another goal is also to facilitate communication between ants and the colony. The pheromone trail will evaporate and reduce its power of attraction [26]. The longer it takes an ant to commute through this path, the longer it is for the pheromone to evaporate. In order for the ants to get the optimal path, several processes are needed. The ACS pheromone control method focuses more on developing and utilizing the best historical pathways than the Ant system. There are three main characteristics of ACS: status transition rules, local pheromone update rules, and global pheromone update rules.

E. Status transition rules

The state transition rule that applies to the first ACS is that the ant placed at point t chooses to go to point v. Then it is assigned a random fractional number q where $0 \leq q \leq 1$. q0 is the probability that the ants explored each stage. pk (t, v) is the probability that ant k chooses to move from point t to point v. When $q \leq q0$, the selection of the point to be addressed uses the following equation:

$$Temporary\ (t,u) = [\tau(t,u_i).[\eta(t,u_i)]^{\beta} \quad i = 1,2,3 \ldots n \tag{1}$$

$$v = max\{[\tau(t,u_i).[\eta(t,u_i)]^{\beta}\} \tag{2}$$

whereas if $q > q0$, the following equation is used:

$$v = P_k(t,v) = \frac{[\tau(t,v)].[\eta(t,v)^{\beta}]}{\sum_{i=1}^{n}[\tau(t,u)].[\eta(t,u_i)^{\beta}]} \tag{3}$$

$$\eta(t,u_i) = \frac{1}{distance(t,u_i)} \tag{4}$$

Where η(t, u) is a heuristic function that is chosen as the inverse of the distance between points t and u. τ(t, u) is the value of the pheromone trace at point (t, u). β is a parameter that considers the relative importance of heuristic information. The value for the parameter β is ≥ 0.

F. Local pheromone update rules

The ants' tour for a solution, but the ants also visit the internodes and change the pheromone levels on them by applying local pheromone renewal rules. The following equations are used for local update updates:

$$\tau(t,v) \leftarrow (1-\rho).\tau(t,v) + \rho.\Delta\tau(t,v) \tag{6}$$

$$\Delta\tau(t,v) = \frac{1}{L_{nn}.C} \tag{7}$$

Where L_{nn} is the length of the tour obtained; C is the number of location points; Δ τ is the change in pheromone. ρ is the amount of pheromone evaporation coefficient with a value of 0 to 1. Each ant's path can be different when the pheromone evaporation takes a long time. It is possible to come up with more alternative solutions. Thus, location points that have previously been traversed by ant tourism can be traversed by other ant tourism.

G. Global pheromone Update Rules

Pheromone points are updated by applying global pheromone renewal rules. All tracks are recapitulated and sorted based on the shortest length of the track. Global pheromone renewal was carried out only in the shortest path since the experiment began.

$$\tau(t,v) \leftarrow (1-\rho).\tau(t,v) + \rho.\Delta\tau(t,v) \tag{8}$$

$$\Delta\tau(t,v) = \begin{cases} L_{gb}^{-1} & if(t,v) \in \ best\ route \\ 0 & \end{cases} \tag{9}$$

$\Delta\tau(t, v)$ is $1/L_{gb}$ if the path (t, v) is the best route that has been travelled and otherwise $\Delta\tau(t, v) = 0$. L_{gb} is the length of the best tour globally since the start of the experiment. The global pheromone update is intended to provide more pheromones on shorter tours.

III. Result and Discussion

The reference point is the number of mutant operators. Where one line of code can include several mutants. Projects are taken randomly obtained on the internet. The ACS algorithm requires data that contains the shortest distance between the average scores for each operator mutation. ACS is used to optimize the search for the shortest route. Figure 2 describes the mutation score from the mutant data. The highest value identifies that many mutants were killed in the mutation operator. In the sample program, as many as 30 source codes show that AOIU and COI have the highest average scores.

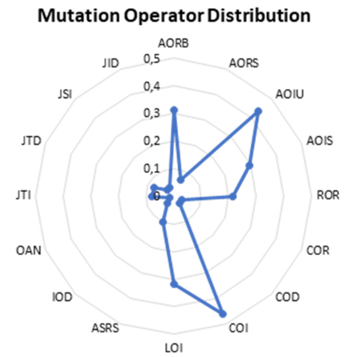

Fig. 2 Mutation Operator Distribution

Figure 2 illustrates the distribution of mutation operators. It shows that COI obtains the highest point. Operators with high scores stated that many were killed during testing. The next calculation is to get the temporary value (t, u) and the probability value based on the starting point (t) to the next untreated point (u). The temporary value is used to determine the points that would be headed next.

$$Probabilitas\ (r,u) = \frac{[r(t,v)].[\eta(t,v)^{\beta}]}{\sum_{i=1}^{n}[r(t,u)].[\eta(t,u_i)^{\beta}]} \tag{10}$$

After completing the calculation process, a probability and accumulative probability is obtained as shown in the table II.

TABLE II
PROBABILITY AND ACCUMULATIVE PROBABILITY

	Probability	**Accumulative probability**
AORB	0,0000000	0,0000000
AORS	0,0000417	0,0000417
AOIU	0,0001758	0,0002175
AOIS	0,0073918	0,0076093
ROR	0,0002554	0,0078647
COR	0,0000324	0,0078971
COD	0,0000324	0,0079295
COI	0,0001175	0,0080470
LOI	0,9916805	0,9997275
ASRS	0,0000556	0,9997831
IOD	0,0000324	0,9998155
OAN	0,0000284	0,9998439
JTI	0,0000457	0,9998896
JTD	0,0000457	0,9999352
JSI	0,0000324	0,9999676
JID	0,0000324	1,0000000

Table II is a table of assistance in recording probability calculations and the accumulated probability that is useful for choosing the next location. The highest probability value as a target location is LOI. Optimization of mutation testing using ACS produces the recommended route in table III.

TABLE III
RECOMMENDATION ACS ROUTE

Mutation Operator	Track
AORB	0,002
LOI	0,020
AOIS	0,082
ROR	0,221
AOIU	0,027
COI	0,362
ASRS	0,022
JTI	0,000
JTD	0,011
AORS	0,033
COR	0,000
COD	0,429
IOD	0,000
JSI	0,000
JID	0,019
OAN	0,300
length of track	1,528

The route of fixing faults is shown in the flow graph in Figure 3. The value of the furthest distance between mutants was obtained from the mutant operator from COR to COD with a value of 0.429. Meanwhile, the shortest distance between mutants is ASRS-JTI, AORS-COR, COD-IOD, and IOD-JSI.

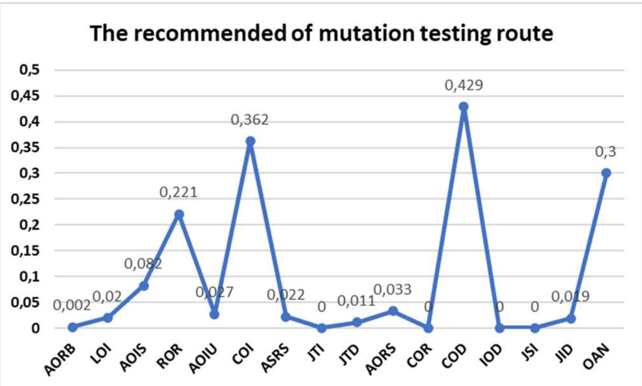

Fig. 3 The Recommended Mutation Testing Route

The comparison between ACS Algorithm and Genetic Algorithm (GA) has a difference of 0.239389. ACS obtains the shortest distance with a value of 1.528254. Meanwhile, the GA trajectory is 1.767643. the comparison of results and paths between ACS and GA is shown in Table IV.

TABLE IV
OPTIMIZATION LENGTH COMPARISON

	ACS	GA
Result	1,528254	1,767643
Path	AORB -> LOI -> AOIS -> ROR -> AOIU -> COI -> ASRS -> JTI -> JTD -> AORS -> COR -> COD -> IOD -> JSI -> JID -> OAN -> AORB	AORS -> AOIU -> AOIS -> ROR -> COD -> COI -> LOI -> ASRS -> JTI -> JSI -> JID

IV. CONCLUSIONS

The effectiveness of fixing faults is an important issue for developers. This paper proposes an optimization using the ant colony system algorithm method to solve the priority problem of very many mutant sequences. This smart method can be immediately applied to software testing. The route selection using the Ant Colony System algorithm resulted in route optimization of 1.528254. Meanwhile, if the genetic algorithm is used, the score is 1.767643. Optimization results are very helpful for developers in improving code in mutation testing. Research states that the best order to handle mutants arises from mutation carriers. The project is selected randomly. Meanwhile, operator mutants that are often killed when mutation testing are AOIU and COI. This research can be further developed with the addition of class-level mutant cases which are produced using class mutation operators. Class mutation operators have different characteristics from traditional mutation operators. In particular, it requires changes to the program structure, such as the definition of class variables.

ACKNOWLEDGMENTS

This research was supported by Universiti Tun Hussein Onn Malaysia (UTHM) and Universitas Putera Batam Indonesia. Computer laboratories are available specifically for the process of researching software testing, and hardware and software can be used to achieve research objectives.

REFERENCES

[1] A. Aghamohammadi, S. H. Mirian-Hosseinabadi, and S. Jalali, "Statement frequency coverage: A code coverage criterion for assessing test suite effectiveness," *Inf. Softw. Technol.*, vol. 129, no. September 2020, p. 106426, 2021.

[2] A. Mustafa, W. M. N. Wan-Kadir, and N. Ibrahim, "Comparative evaluation of the state-of-art requirements-based test case generation approaches," *Int. J. Adv. Sci. Eng. Inf. Technol.*, vol. 7, no. 4–2 Special Issue, pp. 1567–1573, 2017.

[3] F. F. Ismail, R. Razali, and Z. Mansor, "Considerations for cost estimation of software testing outsourcing projects," *Int. J. Adv. Sci. Eng. Inf. Technol.*, vol. 9, no. 1, pp. 142–152, 2019.

[4] P. Delgado-Pérez and F. Chicano, "An experimental and practical study on the equivalent mutant connection: An evolutionary approach," *Inf. Softw. Technol.*, vol. 124, no. April, 2020.

[5] X. Dang, X. Yao, D. Gong, T. Tian, and B. Sun, "Multi-Task Optimization-Based Test Data Generation for Mutation Testing via Relevance of Mutant Branch and Input Variable," *IEEE Access*, vol. 8, pp. 144401–144412, 2020.

[6] P. Pinheiro *et al.*, "Mutating code annotations: An empirical evaluation on Java and C# programs," *Sci. Comput. Program.*, vol. 191, p. 102418, 2020.

[7] N. Gupta, A. Sharma, and M. K. Pachariya, "Multi-objective test suite optimization for detection and localization of software faults," *J. King Saud Univ. - Comput. Inf. Sci.*, no. xxxx, 2020.

[8] A. Usman, N. Ibrahim, and I. A. Salihu, "TEGDroid: Test case generation approach for android apps considering context and GUI events," *Int. J. Adv. Sci. Eng. Inf. Technol.*, vol. 10, no. 1, pp. 16–23, 2020.

[9] S. A. Arnomo and N. Binti Ibrahim, "Priority path for mutant repairs on mutation testing," *Proc. ICAITI 2019 - 2nd Int. Conf. Appl. Inf. Technol. Innov. Explor. Futur. Technol. Appl. Inf. Technol. Innov.*, pp. 71–76, 2019.

[10] J. A. do Prado Lima and S. R. Vergilio, "A systematic mapping study on higher order mutation testing," *J. Syst. Softw.*, vol. 154, pp. 92–109, 2019.

[11] A. V. Pizzoleto, F. C. Ferrari, J. Offutt, L. Fernandes, and M. Ribeiro, "A systematic literature review of techniques and metrics to reduce the cost of mutation testing," *J. Syst. Softw.*, vol. 157, p. 110388, 2019.

[12] A. M. Kazerouni, J. C. Davis, A. Basak, C. A. Shaffer, F. Servant, and S. H. Edwards, "Fast and accurate incremental feedback for students' software tests using selective mutation analysis," *J. Syst. Softw.*, vol. 175, p. 110905, 2021.

[13] X. Yao, G. Zhang, F. Pan, D. Gong, and C. Wei, "Orderly Generation of Test Data via Sorting Mutant Branches Based on Their Dominance Degrees for Weak Mutation Testing," *IEEE Trans. Softw. Eng.*, vol. 5589, no. c, pp. 1–17, 2020.

[14] R. Gheyi *et al.*, "Identifying method-level mutation subsumption relations using Z3," *Inf. Softw. Technol.*, vol. 132, no. April 2020, p. 106496, 2021.

[15] L. Villalobos-Arias, C. Quesada-López, A. Martínez, and M. Jenkins, "Evaluation of a model-based testing platform for Java applications," *IET Softw.*, vol. 14, no. 2, pp. 115–128, 2020.

[16] H. Wang, B. Du, J. He, Y. Liu, and X. Chen, "IETCR: An Information Entropy Based Test Case Reduction Strategy for Mutation-Based Fault Localization," *IEEE Access*, vol. 8, pp. 124297–124310, 2020.

[17] J. M. Zhang, L. Zhang, D. Hao, L. Zhang, and M. Harman, "An empirical comparison of mutant selection assessment metrics," *Proc. - 2019 IEEE 12th Int. Conf. Softw. Testing, Verif. Valid. Work. ICSTW 2019*, pp. 90–101, 2019.

[18] L. Gutierrez-Madronal, A. Garcia-Dominguez, and I. Medina-Bulo, "Combining Evolutionary Mutation Testing with Random Selection," *2020 IEEE Congr. Evol. Comput. CEC 2020 - Conf. Proc.*, 2020.

[19] M. B. Bashir and A. Nadeem, "Improved Genetic Algorithm to Reduce Mutation Testing Cost," *IEEE Access*, vol. 5, no. c, pp. 3657–3674, 2017.

[20] N. Jatana and B. Suri, "Particle Swarm and Genetic Algorithm applied to mutation testing for test data generation: A comparative evaluation," *J. King Saud Univ. - Comput. Inf. Sci.*, vol. 32, no. 4, pp. 514–521, 2020.

[21] M. Nosrati, H. Haghighi, and M. Vahidi Asl, "Test data generation using genetic programming," *Inf. Softw. Technol.*, vol. 130, no. September, p. 106446, 2021.

[22] R. Jangra and R. Kait, "Analysis and comparison among Ant System; Ant Colony System and Max-Min Ant System with different parameters setting," *3rd IEEE Int. Conf.* , pp. 1–4, 2017.

[23] D. N. Mudaliar and N. K. Modi, "Design and Application of m-Mutation Operator in Genetic Algorithm to Solve Traveling Salesman Problem," *8th Int. Conf. Comput. Power, Energy, Inf. Commun. ICCPEIC 2019*, pp. 94–96, 2019.

[24] Q. Zhu, A. Zaidman, and A. Panichella, "How to kill them all: An exploratory study on the impact of code observability on mutation testing," *J. Syst. Softw.*, vol. 173, p. 110864, 2021.

[25] Z. X. Lu, S. Vercammen, and S. Demeyer, "Semi-Automatic Test Case Expansion for Mutation Testing," *VST 2020 - Proc. 2020 IEEE 3rd Int. Work. Validation, Anal. Evol. Softw. Tests*, pp. 1–7, 2020.

[26] N. Yang and Y. Shi, "Research on Tourist Route based on a Novel Ant Colony Optimization Algorithm," *2019 IEEE Int. Conf. Power, Intell. Comput. Syst. ICPICS 2019*, no. 3, pp. 160–163, 2019.