International Journal of Swarm Intelligence Research
Volume 14 • Issue 1

Multi-Objective Optimization Model and Hierarchical Attention Networks for Mutation Testing

Shounak Rushikesh Sugave, Dr. Vishwanath Karad MIT World Peace University, Pune, India*
Yogesh R. Kulkarni, Dr. Vishwanath Karad MIT World Peace University, Pune, India
Balaso and Jagdale, Dr. Vishwanath Karad MIT World Peace University, Pune, India

ABSTRACT

Mutation testing is devised for measuring test suite adequacy by identifying the artificially induced faults in software. This paper presents a novel approach by considering multiobjectives-based optimization. Here, the optimal test suite generation is performed using the proposed water cycle water wave optimization (WCWWO). The best test suites are generated by satisfying the multi-objective factors, such as time of execution, test suite size, mutant score, and mutant reduction rate. The WCWWO is devised by a combination of the water cycle algorithm (WCA) and water wave optimization (WWO). The hierarchical attention network (HAN) is used for classifying the equivalent mutants by utilizing the MutPy tool. Furthermore, the performance of the developed WCWWO+HAN is evaluated in terms of three metrics—mutant score (MS), mutant reduction rate (MRR), and fitness—with the maximal MS of 0.585, higher MRR of 0.397, and maximum fitness of 0.652.

KEYWORDS

Hierarchical Attention Networks, Mutation Testing, Software Testing, Water Cycle Algorithm, Water Wave Optimization

1. INTRODUCTION

Evaluation of software quality is a fundamental research topic in software engineering areas. The exactness of the software application is analyzed using software testing theory (Jatana and Suri, 2020). Software testing acts as a major part of evaluating the software quality or the programs under test (Naeem, et al., 2019). Software testing includes product execution or software application execution along with the group of test cases with the objective of identifying the troubles. The debugging procedure is performed when the testing process shows output variations, which are detected as bugs (Jamil, et al., 2016; Kasurinen, 2010). The test cases are the input sets or the execution preconditions implemented to check whether the particular condition is catered or not (Jovanović, 2006). The

DOI: 10.4018/IJSIR.319714 *Corresponding Author

compilation of test cases accomplished in satisfying the testing conditions are named as the efficient test suite (Chen and Lau, 1998; Jatana and Suri, 2020). Software testers utilize test suites for finding faults in the software when unexpected activity is identified. Hence, the capability of the software testing for identifying the blunders is extremely corresponded to test suites standard. For measuring the quality of the test suites, mutation testing detects whether the test suite is appropriate for identifying the faults by creating syntactic variations in the source code (Jia and Harman, 2010; Naeem, et al., 2019). On the other hand, the code average is also measured to be an efficient technique by analyzing the source code proportion executed by the inputs of the test suite. Nevertheless, the coverage of code alone failed to expose the efficiency in the test suites (Inozemtseva and Holmes, 2014; Naeem, et al., 2019; Kirmani and Wahid, 2015).

Mutation testing is a fault-enabled method that supports producing efficient test cases (Clegg, et al., 2017; Chen and Zhang, 2018; Mao, et al., 2019; Bashir and Nadeem, 2017). It is an efficient fault-driven approach for testing test suite quality by impulsing syntactic blunders. Besides, mutation testing finds whether the test suite is much enough to determine these syntactic faults or not (Jia and Harman, 2010; Singh, et al., 2013; Naeem, et al., 2020). In the process of mutation testing, a large amount of blunder programs, like mutants, which are produced from the real program through the mutant operators. The execution faulty programs are executed besides test suites. If any blunders are founded by the test suite, then the mutants are categorized as killed. Besides, if an outcome generated by the test suite is the same as the real programs, the mutants are regarded as alive. A capability of test suites is computed by the Mutation Score Indicator (MSI), which is a percent of the total number of the killed mutants to overall mutants. The scores produced by mutation testing can assist software testers in locating weaknesses in test suites and also in devising original test cases. With the exclusion of quality assessment of the test suites, mutation testing has also demonstrated its importance in simulating the practical faults (Lou, et al.,2015), faults localization (Moon, et al., 2014), and model transformation testing (Aranega, et al., 2010; Naeem, et al., 2019; Dineva and Atanasova, 2022; Kulkarni, 2022). Moreover, the mutant operators can be devised with respect to the defect model in such a way that the mutant operators generate instances of the recognized flaws or by mutating syntactic workings of the programming language (Wang, et al., 2019; Papadakis, et al., 2019). The latter produces an extremely huge amount of the mutants, thereby making the energy-aware mutation testing impossible as the energy testing should be done on the real devices for achieving appropriate measurements of battery discharge (Jabbarvand and Malek, 2017).

Mutation testing is considered as the most accepted technique in software engineering for verifying software quality beneath tests (Jia and Harman, 2010). In mutation testing, faults are subjected to real programs for generating mutants or the mutated code (Jatana and Suri, 2020). The mutants produced can be excessive in amount and many of these can be trivially killed or can be equivalent (Fraser and Arcuri, 2015; Jatana and Suri, 2020). The objective of the mutation testing enabled the creation of test data for finding test suites that are capable of killing numerous non-equivalent mutants. In general, the overall killed mutants are analyzed using mutation score, which provides a ratio of the killed mutants by test cases to overall mutants (Jia and Harman, 2010; Jatana and Suri, 2020). Mutation testing can be accomplished in the process of software testing at different levels, such as unit, specification level, and incorporation. Mutation testing can be used in several software languages, such as Java, C#, C, AspectJ, SQL, and Fortran (Rani, et al., 2019; Papadakis, et al., 2019; Jatana and Suri, 2020). The mutant operators for several programming languages have progressively evolved during mutation testing research (Jatana and Suri, 2020). The swarm intelligence based optimization algorithms, like Ant Colony Optimization (ACO) and Particle Swarm Optimization (PSO) are used in the generation of test data for mutation testing (Jatana and Suri, 2020; Tiwari, et al., 2012). Genetic algorithm is the popular technique employed in evolutionary testing which utilizes genetical evolution for generating test cases in order to satisfy the test coverage conditions. Furthermore, the Genetic algorithm tries to find the best solution from the input field, and the search is guided by its fundamental element called the fitness function (Bashir and Nadeem, 2017).

The major objective of this paper is to devise a novel approach for the optimal test suite generation with mutation testing based on the proposed WCWWO. Here, the procedure of mutation testing is incorporated with test suites by considering the multi-objectives. In the initial phase, input the sample test programs, and then an optimum test suite is produced on basis of mutation testing using the proposed WCWWO, which is obtained by an amalgamation of WCA and WWO. A process of selecting the best test suite is carried out by considering the multi-objective factors, such as time of execution, test suite size, MRR, and MS. In the Mutation testing phase, HAN is employed to classify the mutants into killed and survived mutants for each test suites.

Contribution of the work:

- An effective WCWWO approach is devised for generating optimal test suites in order to perform the mutation testing. The WCWWO is obtained by an amalgamation of WCA and WWO.
- Here, the multiple objectives, like, time of execution, test suite size, MS, and MRR is considered. Moreover, HAN is employed for classifying the mutants effectively.

The paper is arranged as follows: In section 2, several mutation testing techniques are reviewed and in section 3, developed approaches for mutation testing are illustrated. In Section 4, results and discussions are portrayed and in section 5, the probe paper conclusion is described.

2. MOTIVATION

In this portion, several mutation testing techniques are described with their benefits and drawbacks which support the probers to devise a developed WCWWO+HAN technique for mutation testing.

2.1 Literature Survey

This section reviews and illustrates several mutation testing approaches. Naeem et al. (2020) devised a Machine learning approach for categorizing the mutants. This approach enhanced the mutant detection effectiveness, and also minimized the attempts needed for performing the mutant detection with minimum accuracy loss. However, this technique failed to resolve the issues of various mutation testing domains such as the dominant mutant problem, and the selective mutation problems. Dehmer et al. (2019) introduced Graph similarity measures for the directed and the undirected networks. This measure exhibited that the clustering ability was nearly perfect, but this approach failed to utilize additional properties in order to define effectual and useful graph similarity measures. Bashir and Nadeem (2017) developed an Improved Genetic Algorithm (GA) in order to minimize the cost of mutation testing. This algorithm detects the best test cases in a minimum number of efforts. However, this algorithm does not extend eMuJava v.2 tool for supporting multiple evolutionary techniques in order to perform numerous experiments and assessments. Naeem et al. (2019) developed a Deep learning model for mutation testing. The experimental outcomes are significant with respect to the effectiveness and scalable mutation testing with minimum accuracy loss. However, this technique failed to resolve the issues in numerous mutation testing domains, namely dominator mutant factor and mutant selection by means of machine intelligence.

Jabbarvand and Malek (2017) developed an Energy-Aware Mutation Testing Framework for evaluating the testing quality. For mutation analysis, this framework offers an automated oracle. However, this technique challenges the researchers to implement tests for revealing the defects in energy. Jatana and Suri (2020) introduced an Improved Crow Search Algorithm (ICSA) for the optimal test suite generation. This algorithm achieved maximum detection scores for the Program under Test, but this algorithm failed to improve the effectiveness of test cases. Ma et al. (2018) introduced a Deep Learning System for computing the quality of the test data. This approach effectively measured the test data quality. However, this technique failed to develop advanced mutation operators in order to envelop numerous

aspects of deep learning systems, and also this technique failed to examine the relationships of mutation operators and the operation of mutation operators in initiating the faults equivalent to human faults. Gómez-Abajo et al. (2020) developed a framework, named Wodel-Test model for mutation testing. It designs a generation of MT tools for arbitrary languages. This model failed to apply the technique to additional paradigms languages, such as data flow-enabled or functional.

2.2 Challenges

Several challenges confronted by traditional mutation testing methods are described beneath:

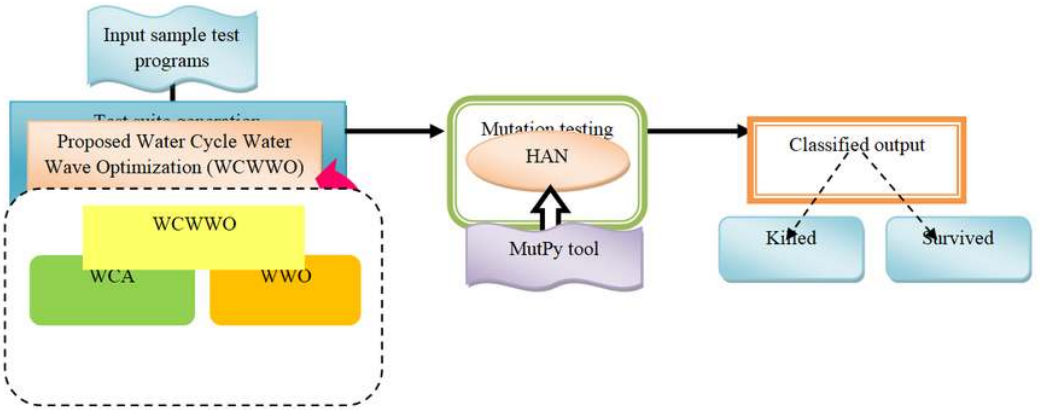
- The creation and execution of huge amount of mutants need of maximum resources and higher costs for computation, and hence the identification of the corresponding mutants becomes highly challenging (Naeem et al., 2020).
- The manual categorization of corresponding mutants consumes more time, and is very tedious. Therefore, the utilization of mutation testing techniques is very rare in the field of the software industry, because of the need of human attempts and running costs (Naeem et al., 2020).
- An improved Genetic algorithm was introduced for mutation testing. However, mutation testing is computationally high, as it needs to execute thousands of mutants (Bashir and Nadeem, 2017).
- In (Naeem et al., 2019), a Deep Learning model was developed for mutation testing, but the mutation testing cost is very high as it requires the generation and execution of individual mutants with the test suites.
- A vital challenge with the process of mutation testing is an oracle issue in evaluating, whether test case implementation kills the mutants or not. This is mainly a challenge with the process of energy testing, as the state-of-the-practice is typically a manual procedure where the engineer determines the power trace of running a test for finding the energy inefficiencies that might result in identifying the defects (Jabbarvand and Malek, 2017).

3. PROPOSED MULTI-OBJECTIVE BASED OPTIMIZATION MODEL FOR MUTATION TESTING

Mutation testing is generally employed for determining test suite quality for supporting software testers in order to enhance the quality. Mutation testing is considered as a fault-driven technique, and the concepts associated with it are illustrated as follows. The mutant is generated by varying the original program syntax. A rule used for performing the variations in syntax is known as the mutant operator. If the test datum can differentiate the output among the mutant and the real program, hence mutant is regarded as killed. On other hand, if the mutant is equivalent, then it is not killed by any kind of test datum. This paper presents a multi-objective-based optimization approach for mutation testing. Here, the generation of optimal best suites is carried out using the proposed WCWWO where the mutation testing is combined. The various steps involved in generating the optimal test suites with the process of mutation testing are, inputting the sample test programs, test suite generation by means of hybrid optimization, and mutation testing. In the initial phase, input the sample test programs and the optimal test suites are chosen based on the mutation testing using the newly designed optimization algorithm named WCWWO, which is derived by the integration of WCA (Eskandar, et al., 2012), and WWO (Zheng, 2015), respectively. The proposed WCWWO is utilized for developing and detecting the finest test suites which satisfy multi-objectives, such as time of execution, test suite size, MS, and MRR. In the mutation testing stage, the MutPy tool is employed for the generation of mutants such that the HAN (Li, et al., 2019) is utilized for categorizing the equivalent mutants based on the graph-based similarity metrics. Moreover, the mutant score, killed and survived mutants will be identified for every test suite based on the MutPy tool. Figure 1 presents the schematic representation of mutation testing with the proposed WCWWO.

Let us consider a program database, D with a set of input programs, and is represented as,

Figure 1.
Diagrammatic representation of mutation testing using developed WCWWO



$$D = \left\{I_1, I_2, ..., I_e, ..., I_f\right\} \tag{1}$$

Where, e^{th} input program is signified by I_e and f represents the total input programs

Consider an input program under test denoted as I and the test suite specified as T. Here, the set of selected mutants is generated by the selection of the mutant operator M, and N is a set of non-equivalent mutants. Moreover, random test suites e are created with different sizes that are $\left\{T_1, T_2, ..., T_e\right\}$ and the group of mutants is killed by T_i such that $1 \le i \le e$, and is represented as $B\left(T_i, M\right)$.

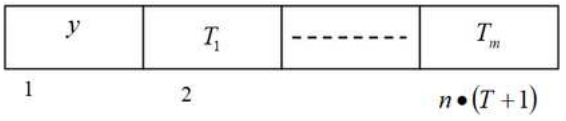
3.1 Test Suite Generation Using Proposed WCWWO with Mutation Testing

In recent days, mutation testing has gained attraction towards correcting errors in the software. The major aim of the developed technique is to enhance the process based on the bio-inspired algorithmic techniques for finding the mutants from the codes. Here, the mutation testing concept is used in generating the optimal test suites in such a way that the faults are determined in the software below test. The process of generating the test suites is performed using the developed WCWWO, which is newly designed by incorporating WCA (Eskandar, et al., 2012), and WWO (Zheng, 2015). In addition, the fitness function, solution encoding, and algorithmic phases of the proposed WCWWO are illustrated below,

3.1.1 Solution Encoding

Solution encoding is used for identifying the solution of the developed WCWWO algorithm in a better way. The solution vector is computed randomly based on the test suite generation. Figure 2 interprets the solution encoding of the proposed WCWWO.where, n represents the total input variables for test programs, T denotes the maximum test cases allowed, and $n \bullet \left(T+1\right)$ indicates the overall

Figure 2.
Solution encoding of proposed WCWWO



test cases. Here, the initial element indicates the total test cases to be chosen, such that $1 \leq y \leq T$ and the second element to the final element represents the test cases.

3.1.2 Fitness Function

When the test suite obtains all the accessible feature settings, then the test suite is chosen, whereas it is discarded when the test suite failed to accomplish the full coverage. The fitness value can be computed by considering the maximal value of chosen test cases. The optimal acquisition of the new solution arises when there exists a violation. The equation of fitness measure is represented as,

$$\left(max() \frac{1}{4}\left[\left(1-E\right)+\left(1-S\right)+MR+MS\right]\right) \tag{2}$$

where, MS signifies mutant score, MR represents the mutant reduction rate, S denotes the test suite size, and E indicates the execution time of the program in seconds. Here, the fitness value is termed as the maximization function.

(i) Mutant score

The mutant score is a ratio of the percentage with the mutants killed to the overall mutants. If the mutation score achieved is 100%, then test cases are said to be mutation adequate. The equation for the mutation score is formulated as,

$$MS = \frac{G_m}{T_m - E_m} \tag{3}$$

where, G_m denotes the dead mutant, T_m denotes the total mutant, and E_m specifies the equivalent mutant.

(ii) Mutant reduction rate

The mutant reduction rate helps in resolving the problems based on computational cost. During software testing, a higher number of mutants may increase the maximum mutation testing cost, and this can be efficiently solved by the mutation reduction rate parameter. The equation for MRR is formulated as,

$$MRR = \left[1 - \frac{M_g}{T_h}\right] \tag{4}$$

where, M_g represents the mutant generated, and T_h denotes the threshold.

(iii) Test suite size

The test suite size is evaluated based on the overall test cases and function calls of non-library during the execution time. The test suite size is expressed as follows,

$$S = \sum_{i=1}^{n} \frac{Z_i}{n} \tag{5}$$

where, z_i represents the size of the test case i and n represents the total input variables for test programs.

3.1.3 Algorithmic Steps of Developed WCWWO Algorithm

The generation of the best test suite is carried out using the proposed WCWWO, which is designed by the integration of WCA (Eskandar, et al., 2012), and WWO (Zheng, 2015). WCA (Eskandar, et al., 2012) is a nature-based algorithm for addressing optimization issues. WCA works on the basis of water cycle procedure and measured the flow of rivers and streams into the sea. This theory begins with the initial population named as raindrops, thus the finest raindrop is considered as sea. Subsequently, the total amount of the finest raindrops is selected as the river, whereas residual raindrops are selected as streams passing to rivers and sea. On the other hand, WWO (Zheng, 2015) is a meta-heuristic approach inspired by the shallow water wave process in order to resolve global optimization problems. The incorporation of WCA with the WWO minimized the computational issues, and achieved optimal test suites. The algorithmic steps of the developed WCWWO are portrayed below as,

Initialization: A population is initialized with H number of raindrops and is denoted by,

$$Q = \left\{Q_1, Q_2, \cdots, Q_d, \cdots Q_H\right\}; 1 \le d \le H \tag{6}$$

Here, the total amount of raindrop solutions is signified by H and Q_d represents the d^{th} raindrop.

ii) ***Compute fitness measure:*** A fitness measure is utilized for identifying an optimum solution by computing the best fitness value, and the fitness measure is illustrated in equation (2).

Compute cost function: The decisive variables $\left(Q_1, Q_2, \ldots, Q_{H_{var}}()\right)$ are indicated as values of floating point or these variables are denoted as prearranged set for continuous and discrete limitations. A raindrop cost is determined using a computation of the cost function, and is expressed as,

$$A_l = Cost_l = k\left(Q_1^l, Q_2^l, \ldots, Q_{H_{var}}^l()_{pop}\right) \tag{7}$$

Here, total number of raindrops is denoted by H_{pop} and an overall design value is signified by H_{var}. At the initial phase, H_{pop} raindrops are produced. Overall H_{sr} from optimal minimum values are considered as the rivers and sea. Additionally, H_{sr} signifies a summation of total rivers and an individual sea.

$$H_{sr} = \Re + \underbrace{1}_{sea} \tag{8}$$

where, $\Re$ denotes the total number of rivers. On the other hand, the residual raindrops develop streams passing to sea or rivers, and an equation is expressed by,

$$H_{Raindrops} = H_{pop} - H_{sr} \tag{9}$$

Compute an intensity of flow: Raindrops can be allotted to sea and river with respect to flow intensity and hence an equation for computing the flow intensity is formulated as,

$$H_t = round\left\{\left|\frac{Cost_t}{\sum_{l=1}^{H_{sr}} Cost_l}\right| \times H_{Raindrops}\right\} t = 1,2,....H_{sr} \tag{10}$$

Where, H_t denotes the streams passing to the particular rivers or sea.

(v) ***Evaluate a stream flow into sea or river:*** The streams are produced from every raindrop, afterward merged together to produce the novel rivers. Furthermore, streams straightly move into the sea and then all streams and the rivers merge in a sea.

The stream passes to rivers other than the fusion line amid them based on preferred distance irregularly, and an equation illustrated is,

$$Q \in \left(0, A \times d\right), A > 1 \tag{11}$$

Where, a value A lies between range 1 and 2, and term d signifies a current distance among rivers and streams. The value of A being larger than 1 facilitate streams to pass into the rivers in different directions. The rule can be employed in passing rivers to the sea. Thus, rivers and streams with the new position are expressed as,

$$Q_{stream}^{l+1} = Q_{stream}^{l} + rand \times A \times \left(Q_{river}^{l} - Q_{stream}^{l}\right) \tag{12}$$
$$Q_{river}^{l+1} = Q_{river}^{l} + rand \times A \times \left(Q_{sea}^{l} - Q_{river}^{l}\right) \tag{13}$$
$$Q_{river}^{l+1} = Q_{river}^{l}\left(1 - randA\right) + randA \times Q_{sea}^{l} \tag{14}$$

By combining WWO with WCA, optimization problems in WCA can be minimized, and the standard equation of WWO is represented in terms of WCA as,

$$Q_{river}^{l+1} = Q_{river}^{l} + rand\left(-1,1\right)\lambda Y_g \tag{15}$$
$$Q_{river}^{l} = Q_{river}^{l+1} - rand\left(-1,1\right)\lambda Y_g \tag{16}$$

By substituting equation (16) in equation (14)

$$Q_{river}^{l+1} = \left(Q_{river}^{l+1} - rand\left(-1,1\right)\lambda Y_g\right)\left(1 - randA\right) \times randA\, Q_{sea}^{l} \tag{17}$$
$$Q_{river}^{l+1} = Q_{river}^{l+1}\left(1 - randA\right) - rand\left(-1,1\right)\lambda Y_g\left(1 - randA\right) + randA\, Q_{sea}^{l} \tag{18}$$
$$Q_{river}^{l+1} - Q_{river}^{l+1}\left(1 - randA\right) = randA\, Q_{sea}^{l} - rand\left(-1,1\right)\lambda Y_g\left(1 - randA\right) \tag{19}$$

$$Q_{river}^{l+1}\left(1-1+randA\right)=randA\,Q_{sea}^{l}-rand\left(-1,1\right)\lambda Y_{g}\left(1-randA\right) \tag{20}$$

$$Q_{river}^{l+1}=\frac{randA\,Q_{sea}^{l}-rand\left(-1,1\right)\lambda Y_{g}\left(1-randA\right)}{randA} \tag{21}$$

$$Q_{river}^{l+1}=Q_{sea}^{l}-rand\left(-1,1\right)\lambda Y_{g}\left\{\frac{1-randA}{randA}\right\} \tag{22}$$

Where, $rand$ signifies a random number uniformly distributed within the range between 0 and 1, value A lies between a range 1 and 2, Y_g denotes the length of g^{th} dimension search space, Q_{sea}^{l} specifies optimal solution, and λ has a value of 0.5

Estimate the condition for evaporation: Evaporation is a significant factor that protects a technique from immature convergence. Evaporated water is absorbed by the atmosphere, thereby forming clouds afterward it gets concise in the colder part of the atmosphere. Hence, the evaporated water is released as rain to the earth. The rain produces newer streams passing to the rivers and thereafter to sea. Furthermore, d_{max} signifies the lowest number nearer to 0, and the d_{max} value reduces adaptively as,

$$d_{max}^{l+1}{}^{\frac{d_{max}^{l}}{\kappa}} \tag{23}$$

where, κ signifies the maximum iterations.

Raining process: At once the evaporation process gets finished, the raining process is initialized. Here, a neo raindrop generates streams in different areas, and an equation for representing the newer location of freshly produced streams is expressed by,

$$Q_{stream}^{new}=wx+rand\times\left(kx-wx\right) \tag{24}$$

Here, the lower bound is denoted by wx and kx represents the upper bounds.

$$Q_{stream}^{new}=Q_{sea}+\sqrt{\mu}randt\left(1,H_{var}()\right) \tag{25}$$

where, μ specifies the coefficient which exhibits a limit of the searching area close to the sea, and a value is set to 0.1, $randt$ signifies a random number usually distributed.

viii) ***Feasibility evaluation:*** The feasibility of the solution is evaluated in order to discover the best value. If a freshly obtained solution has an optimal value than the existing value, then a solution is renewed with a newly achieved optimal value.

ix) ***Termination:*** The above illustrated steps are carried out in an iterative manner till the finest solution is attained. A pseudo-code of the proposed WCWWO is described in algorithm 1.

A proposed optimization algorithm, named WCWWO with mutation testing is very effective in generating the optimal test suites.

3.2 Mutant classification using HAN

An equivalent mutant does not have any impact on program execution. The mutational impacts can be accessed by validating the program state at the end of the operation. In general, one can access

Algorithm 1.
Pseudo code of proposed WCWWO

Pseudo code of proposed WCWWO
Start
Initialize the parameters H_{sr}, d_{max}, H_{pop}
Generate the initial population using equation (6) randomly
Compute the cost of each raindrop by equation (7)
Compute flow intensity by equation (10)
Compute a flow of the stream to rivers by equation (11)
for
if $A > 1$
Satisfy evaporation condition
else
Replace the position of the river with sea
end if
if $\left\| H_{sea}^{t} - H_{river}^{t} \right\| < d_{max}$
Replace a locale of the river with sea
else
Satisfy evaporation criteria
end if
Begin a process of raining by equation (24) and equation (25)
Minimalize d_{max} by equation (23)
Validate convergence criteria
Satisfy convergence criteria
Re-iterate a process
end for
end

the mutation impacts even though the assessment is not complete. In the same way, one can calculate the variations of the program characteristics among the original and the mutant version. Moreover, control flow is the factor, which is particularly easy to compute. If the mutation varies the control flow performance, the different syntax is implemented in a different order, which may lead to non-

equivalence. Several steps are included in order to calculate whether the mutant program and the input program are equivalent or not. Initially, the mutant program, and the input program are considered as input in a simultaneous way. After that, the graph extraction is carried out based on the input program and the mutant program. Once the graph extraction is performed, entropy calculation is done for the extracted graphs of both the input program and output program. Thereafter, the similarity among the entropies is matched, and then the features are extracted. The features extracted are subjected to the HAN classifier (Li, et al., 2019), and finally the equivalent mutants are classified. Figure 3 illustrates the schematic representation of the equivalent mutant classification model using HAN.

3.2.1. Graph Construction

The initial phase in the mutant classification is graph construction. Here, the graphs are extracted based on the input program and the output program in such a way that the graphical program representations are obtained, which involves CFGs, PDGs, and ASTs using software source code. In constructing the graphs, the source code files are read and considered as the input. The input produces various graphical program representations and exports it to various file formats of graphs that include DOT, GML, and JSON. Furthermore, it focuses on extracting the graphical program representations, and then to perform the further analysis.

Figure 4 shows the graph instance. From the graph, it is illustrated that three different events, such as (a-b), (a-c), and (b-d) are obtained, and are mathematically expressed as, $\left(\frac{2}{4}\right),\left(\frac{1}{4}\right),\left(\frac{1}{4}\right)$.

Figure 3.
Schematic representation of mutant classification using HAN

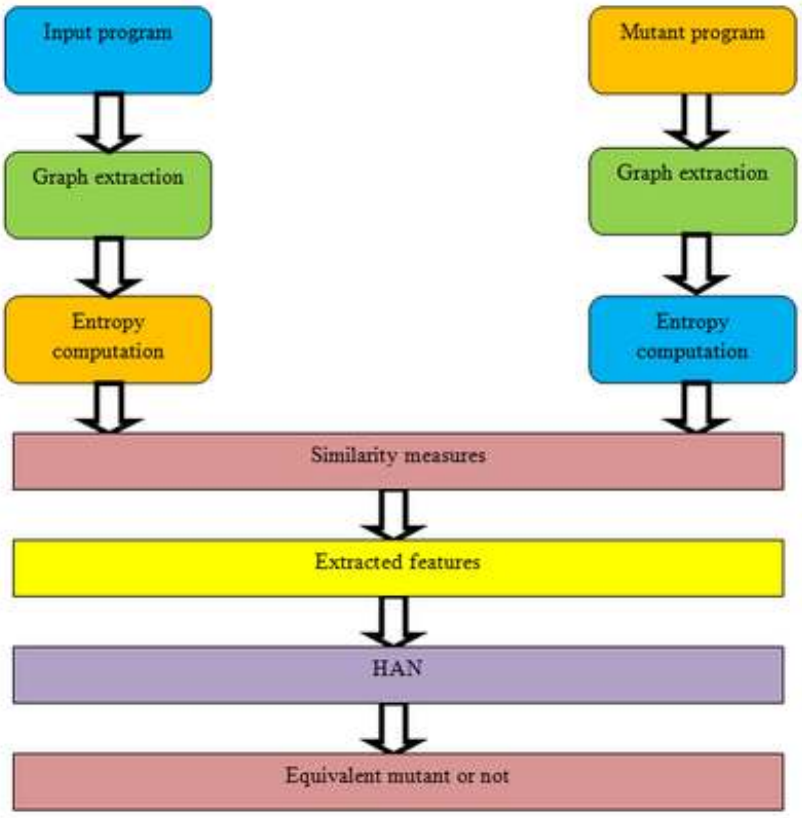
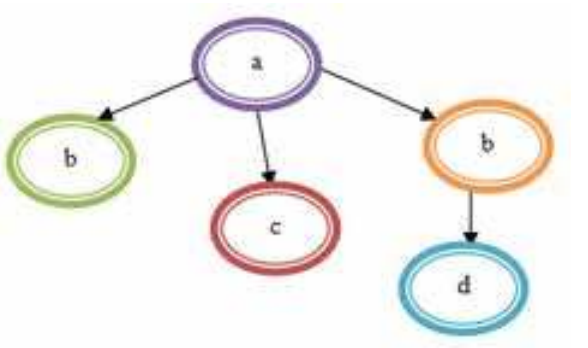


Figure 4. Instance of graph



3.2.2. Entropy Computation

Entropy (Naeem, et al., 2019) is a typical measure, which is used to compute the ambiguity in several data and is used for enhancing the mutual information from various operations. The higher accessibility of entropy changes encouraged appropriate preference for specific operations. Therefore, the data entropy is utilized for targeting the difference between the groups. The simpler utilization of mutant operators is used for overcoming the problem that mutant operators do not identify premature convergence. Here, the mutant features are defined on the basis of information entropy. Every variable is defined on the basis of mutant operators, and is manipulated as cluster of random variables comprising of specific information. When the mutant operator produces syntax error in the program, a portion of information becomes modified, whereas the remaining portion of the information becomes protected in such a way that more information is retained. Information entropy is used as a mutant operator feature, which utilizes the preserved mutation information.

Let us consider a discrete random variable b with a possible events, expressed as $B = \left\{b_1, b_2, \ldots, b_a\right\}$, where the event probability b_1 occurrence is $p\left(b_1\right)$, then the information of the event b_1 is represented as,

$$M\left(b_h\right) - \log q\left(b_h\right) \tag{26}$$

where, $M\left(b_1\right)$ signifies the uncertainty reduction of a random variable B with respect to the event occurrence b_1. The entropy measured for the above graph is expressed as,

$$Entropy = -\sum\left[\frac{2}{4}\log\frac{2}{4} + \frac{1}{4}\log\frac{1}{4} + \frac{1}{4}\log\frac{1}{4}\right] \tag{27}$$

3.2.3. Feature Extraction using Similarity Measures

Here, the graph similarity (Dehmer, et al., 2019) and the distance measures are designed. These measures are designed based on the concept of network mapping with the positive real number and provide values, which can be compared. The similarity measures are used for calculating the graph similarity. This technique provides a basis for the efficient construction of the graph similarity measures. The generated values are used to compute the structural similarity of the different graph pairs. Moreover, the process is initialized by representing four different measures on the positive real

numbers and poses the similarity measure conditions. Let us consider $u, v \in S$ and $\beta, \alpha > 1$, then the similarity measures are represented as,

$$r_1\left(u,v\right) = \frac{1}{1+\left|u-v\right|^{\beta}} \tag{28}$$

$$r_2\left(u,v\right) = \frac{1-\left|u-v\right|^{\beta}}{1+\left|u-v\right|^{\beta}} \tag{29}$$

$$r_2\left(u,v\right) = \frac{1-\left|u^{\alpha}-v^{\alpha}\right|^{\beta}}{1+\left|u^{\alpha}-v^{\alpha}\right|^{\beta}} \tag{30}$$

$$r_2\left(u,v\right) = \frac{1-\left|u-v\right|^{\beta}-\left|u^{\alpha}-v^{\alpha}\right|^{\beta}}{1+\left|u-v\right|^{\beta}\left|u^{\alpha}-v^{\alpha}\right|^{\beta}} \tag{31}$$

Assume $r_1\left(u,v\right)$. If $u = v$, then $r_1\left(u,v\right) = 1$. Also, if $r_1\left(u,v\right) = 1$, then $u = v$. In addition, the symmetry property is satisfied as $\left|u-v\right|\beta = \left|v-u\right|\beta$. Also assume $r_1\left(u,v\right) \geq 0$, then it results in inequality $1+\left|u-v\right|\beta > 1$, which satisfies $u, v \in S$. Thus, the similarity measure condition for $r_1\left(u,v\right)$ is illustrated in the above-expressed equations. Finally, the extracted features are denoted as F.

3.2.4. Equivalent Mutant Classification using HAN Classifier

The equivalent mutant classification is performed by verifying the program state at the end of the operation as the tests perform. Here, extracted feature output F is represented as input for equivalent mutant classification, which is performed using the HAN classifier. The individuals can be able to calculate the variation in the program characteristics among the mutant and the original version. HAN classifier (Li, et al., 2019) is utilized for calculating whether the input program and mutant program are equivalent or not. The benefit of employing the HAN classifier is, it attains optimal solutions with lesser computational time and cost, and therefore HAN classifier is accomplished for effectual equivalent mutant classification. The architectural steps of HAN are explained beneath,

i) Architecture of HAN

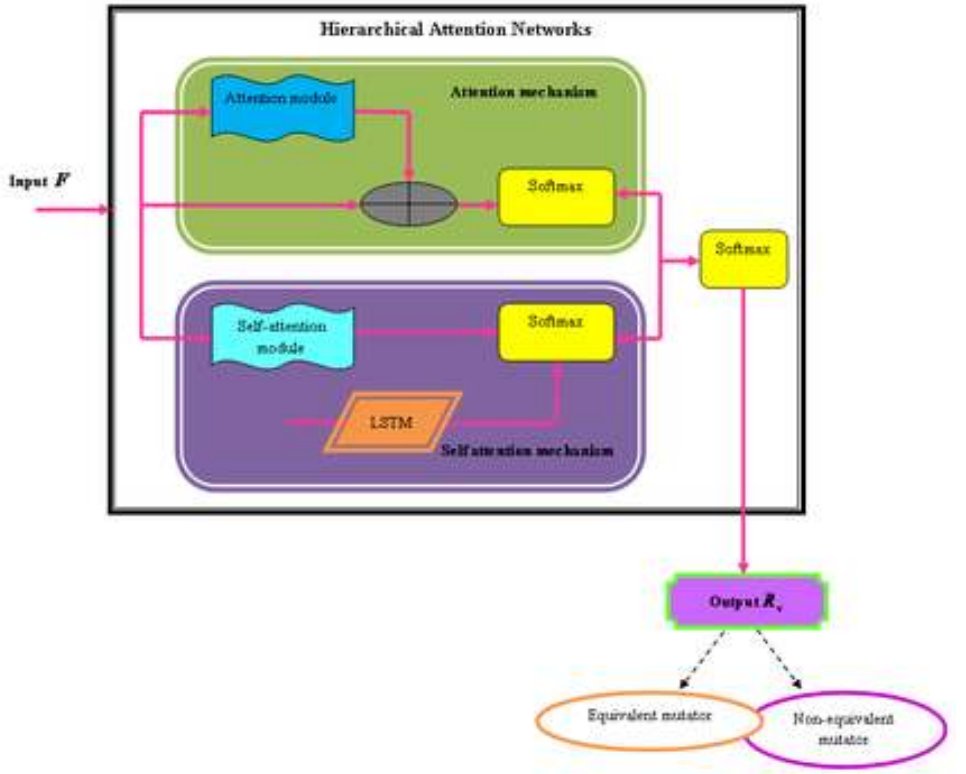
An architectural illustration of HAN is portrayed in figure 5. The overall structure of HAN consists of numerous units, namely the attention module, softmax, long short-term memory (LSTM), and self-attention module. A model employs an extracted feature output F wherein the HAN classifier is utilized to perform the mutation testing.

ii) Attention model

The attention module includes three different modules, namely CNN, LSTM, and attention layer

CNN: In the CNN module, feature layers of VGGNet are used for the extraction of feature maps. An initial procedure is done for image rescaling in 448×448 pixels. Thus, an outcome attained from feature layer VGGNet is in the size of $512 \times 14 \times 14$. In addition, the 512×196 dimension vector is stationed in a fully connected layer with respect to $tanh$ function that transmutes it to dimension vector having size 1024×196

Figure 5.
Structure of HAN



$$J_{ji} = \tan h(O_i J_i + s_i) \tag{32}$$

where, J_i signifies the feature vector of the overall region, J_{ji} signifies every region. An extension of dimension makes an incorporation procedure in-depth.

LSTM: LSTM includes different memory cells, therefore consists of four various stages in renewing cell states. A starting stage creates a decision for evaluating information to be discarded from the cell state, whereas other stages create the decision based on the novel information which needs to be saved in the cell state, and hence equation is formulated by.

$$y_m = \sigma\left(O_y\left[a_{m-1}, w_m\right] + s_y\right) \tag{33}$$
$$i_m = \sigma\left(O_i\left[a_{m-1}, w_m\right] + s_i\right) \tag{34}$$
$$U_j = \tan h\left(L_U.\left[a_{m-1}, w_m\right] + s_U\right) \tag{35}$$

where, U_j signifies a memory to be known, w_m denotes an input vector, a_m represents hidden state, forget gate is signified by y_m, i_m denotes an input gate.

An updation of the older state U_{m-1} to neo state U_m is specified by,

$$U_m = y_m * U_{m-1} + i_m * U_i \tag{36}$$
$$R_m = \sigma\left(O_R.\left(a_{m-1}, w_m\right) + s_R\right) \tag{37}$$
$$a_m = R_m * \tan h(U_m) \tag{38}$$

where, R_m signifies the output gate.

Attention layer: Initially, O_ρ as well as O_κ are subjected to fully connected layer and then incorporated with $\tanh$ function. An attention distribution map is obtained with respect to the softmax function,

$$a_{att} = \tan h(\left(O_{\kappa,att}.J_{ji} + s_{ji,att}\right) \oplus \left(O_\rho + s_{\rho,att}\right)) \tag{39}$$
$$C_\xi = Soft\max(O_q.a_{att} + s_q) \tag{40}$$

Here, O_κ is 196×1024 represents the dimension matrix, O_ρ indicates 1024 dimension matrix, $O_{\kappa,att}$ and $O_{\rho,att}$ signify the 1024×512 dimension matrix, O_q indicates 512 dimension vector and C_ξ specifies a shape as well as addition vector is represented by $\oplus$. A weighted sum is computed on basis of attention distribution map and formulated by,

$$J_i^{'} = \sum q_i, J_{ji} \tag{41}$$

iii) Self-Attention model

A self-attention model is employed for collecting global information. It is expressed by,

$$\varepsilon_i = w_i + \sum_{p=1}^{N_q} \frac{c\left(g_i, n_i\right)}{\delta}\left(O_u, g_p\right) \tag{42}$$

where, $c\left(.\right)$ represents the function between i and p, linear transform is signified by O_u and δ denotes the normalization factor.

Therefore, an output developed from HAN is indicated by R_τ which signifies the classified output is equivalent mutant or not.

3.3. Mutant Score Computation Model

The major objective in mutation testing is to validate whether an available test suite discovers mutant. For this reason, the test suites are executed on these mutated versions of the input program. If one test run fails, then it is indicated that the mutant remains killed or detected, else mutant is alive. If test suites have the capability to discover numerous mutants, then the test suite is said to be more efficient than the other test suites. This is computed by the term, named mutation score, and the various processes involved in mutation score computation are illustrated below as follows,

i) Test suite and input program as input

Initially, the input is considered, wherein the data is presented to the program. Here, the input considered is the sample program, which can offer a set of built-in-functions in order to output the data on the screen, and save it in binary or test files.

ii) Mutant generation

In the process of mutant generation, numerous program variants are formed by the set of rules also called as the mutant operators, which creates only one syntax variation at a particular time, thereby generating only one mutant. As a result, the overall produced mutants are executed in the test suites under computation.

iii) Equivalent mutant classification

If the real program and mutant program are equivalent, then the mutant is considered to be identical and no test case can kill it. Equivalent mutants are considered as an issue in the process of mutation testing as the equivalence is not decidable, and hence it is very difficult to identify if the un-killed mutant is killable. Furthermore, the mutant is said to be equivalent when no such test case differentiates the original program output and the mutant output.

iv) Computation of mutant score

The mutant score is indicated as a ratio of the percentage of killed mutants to overall mutants. If the mutation score is 100%, then test cases are said to be mutation adequate.

3.4. Detailed Flow Chart of the Process

The major aim of the research is to design an approach for mutation testing by considering multi-objective driven optimization model. Here, the optimal test suite generation is carried out with the developed WCWWO where the mutation testing is incorporated. The various phases involved in generating optimal test suites with mutation testing are described. Initially, the input test suite solution and the input programs are fed as an input at the equivalent time. Based on the test suite solution and input programs, the mutant generation is performed to generate the mutants. Once the mutants are generated, it is presented to the equivalent mutant classification, which is carried out using HAN (Li, et al., 2019). In addition, the test suites with the fitness measures, such as execution time, size of the test suite, MS, and MRR are considered. Thereafter, killed, and the survived mutants are classified for every test suite. Furthermore, the proposed optimization algorithm, named WCWWO is employed for generating the finest test suites. Once the test suites are generated, the process gets repeated from the mutant generation step in order to find the optimal test suites. Figure 6 presents the detailed flow chart of the proposed WCWWO technique for mutant testing.

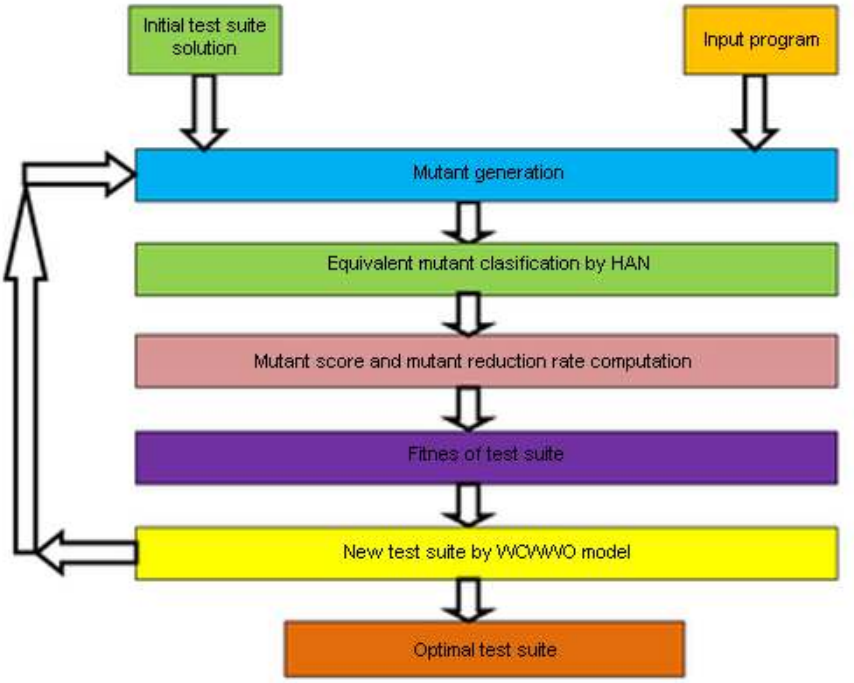
4. RESULTS AND DISCUSSION

This portion illustrates the efficiency of the proposed WCWWO+HAN using performance metrics, such as MS, MRR, and fitness. The analysis of the method is carried out by varying the test case sizes.

4.1. Experimental Set-Up

The experimentation of the developed WCWWO+HAN technique is performed in Python tool by Python Interpreters Benchmarks dataset (https://pybenchmarks.org/) having PC with Windows 10 OS, 2GB RAM, and Intel i3 core processor. Here, the experimentation is done with epochs=40 and batch size=50.

Figure 6.
Flow chart of the proposed WCWWO+HAN method for mutant testing



4.2. Dataset Description

The dataset employed for the analysis is Python Interpreters Benchmarks for improving the test programs (https://pybenchmarks.org/). Here, the three interpreters taken for the analysis from the Python Interpreters Benchmarks are Fibonacci code (https://pybenchmarks.org/u64q/benchmark.php?accessedtest=fibonacci&lang=python3&id=3&data=u64q), Fankuch redux code (https://pybenchmarks.org/u64q/benchmark.php?test=fannkuchredux&lang=python3&id=3&data=u64q), Spectral norm code (https://pybenchmarks.org/u64q/benchmark.php?test=spectralnorm&lang=python3&id=3&data=u64q), and meteor-contest benchmark (Kulkarni, 2022).

4.3. Evaluation Measures

The performance assessment is carried out by the evaluation metrics, such as MS, MRR, and fitness, and is illustrated in section 3.1.2.

4.4. Comparative Methods

The various comparative techniques used for the assessment are Machine Learning (Naeem, et al., 2020), Improved GA (Bashir and Nadeem, 2017), and Deep Learning (Naeem, et al., 2019).

4.5. Comparative Analysis

An assessment of approaches with MS, MRR as well as fitness is performed by differing test case sizes from 100 to 400 and the analysis is carried out using three interpreters, namely Fibonacci code, Fankuch-redux code, and Spectral norm code.

4.5.1 Analysis Based on Fibonacci Code

Figure 7 represents an analysis on basis of Fibonacci code with respect to MS, MRR, and fitness by varying the test case sizes. Figure 7a) portrays the analysis based on MS. For the test case size 100, the MS value obtained by Machine learning, Improved GA, Deep learning, and proposed WCWWO+HAN are 0.478, 0.508, 0.546, and 0.571. The assessment of MRR is presented in figure 7b). Considering test case size 300, the proposed WCWWO+HAN achieved MRR value of 0.366, where the MRR value obtained by current approaches, like Machine learning is 0.310, Improved GA is 0.344, and Deep learning is 0.359. Figure 7c) presents the analysis based on fitness. The developed WCWWO+HAN obtained a fitness value of 0.627, while current methods, like Machine learning, Improved GA, and Deep learning achieved a fitness value of 0.361, 0.580, and 0.622 for the test case size 200.

4.5.2 Analysis using Fankuch-Redux Code

The analysis based on Fankuch-redux code by varying the test case sizes with regarding to performance metrics, namely MS, MRR, and fitness is portrayed in figure 8. Figure 8a) represents an analysis based upon MS. The MS value achieved by the Machine learning technique is 0.518, Improved GA is 0.534, Deep Learning is 0.552, and the proposed WCWWO+HAN is 0.572 for the test case size 300. The assessment based on MRR is depicted in figure 8b). When test case size is 200, MRR value obtained by the proposed WCWWO+HAN is 0.381, while current approaches, like machine learning, Improved GA, and Deep learning achieved an MRR value of 0.194, 0.340, and 0.357. The fitness analysis is shown in figure 8c). For the test case size 100, the fitness value measured by the machine learning, Improved GA, Deep learning, and proposed WCWWO+HAN is 0.424, 0.559, 0.585, and 0.619.

4.5.3 Analysis Based on Spectral Norm Code

Figure 9 represents an estimation utilizing spectral norm code by considering metrics, such as MS, MRR, and Fitness. Figure 9a) shows the assessment based on MS. The MS value computed by developed WCWWO+HAN is 0.563, while MS value computed by current approaches, like machine

Figure 7.
Analysis of proposed WCWWO+HAN utilizing Fibonacci code a) MS, b) MRR, and c) Fitness

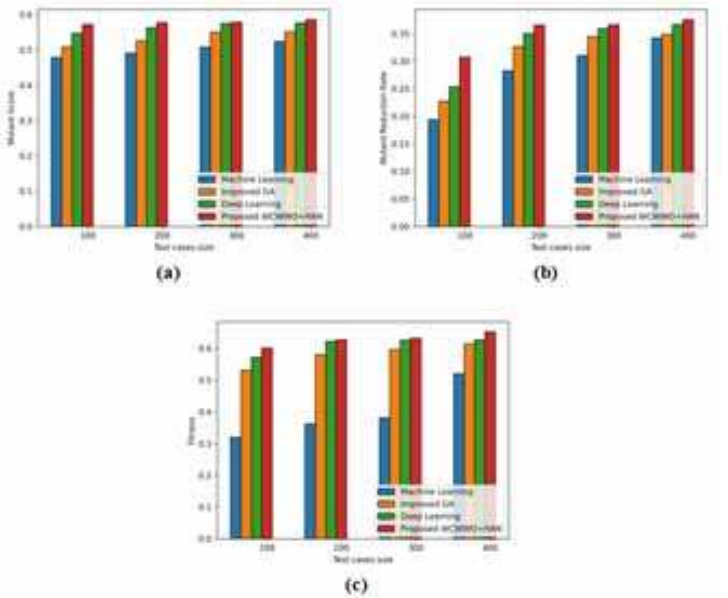
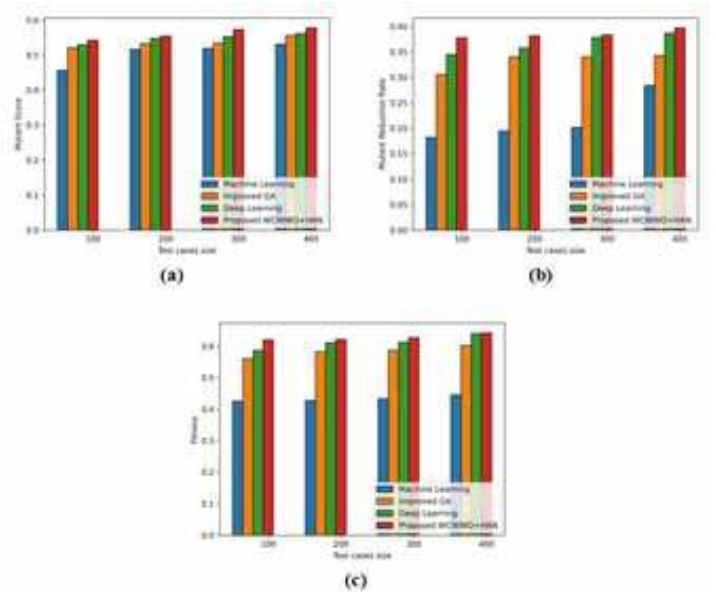


Figure 8.
Analysis of the proposed WCWWO+HAN using Fankuch-redux code a) MS, b) MRR, and c) Fitness



learning is 0.446, Improved GA is 0.499, and Deep learning is 0.530 for the test case size 100. The assessment of MRR metric is depicted in figure 9b). When test case size is 300, MRR value calibrated by the machine learning approach is 0.281, Improved GA is 0.328, Deep Learning is 0.353, and proposed WCWWO+HAN is 0.368. The analysis using fitness is depicted in figure 9c). For the test case size 200, the fitness value obtained by the Machine learning, Improved GA, Deep learning, and proposed WCWWO+HAN are 0.376, 0.588, 0.604, and 0.612.

4.5.4 Analysis Based on Meteor-Contest Benchmark

Figure 10 represents an estimation utilizing meteor-contest benchmark by considering metrics, such as MS, MRR, and Fitness. Figure 10a) shows the assessment based on MS. The MS value computed by developed WCWWO+HAN is 0.522, while MS value computed by current approaches, like machine learning is 0.446, Improved GA is 0.476, and Deep learning is 0.496 for the test case size 100. The assessment of MRR metric is depicted in figure 10b). When test case size is 300, MRR value calibrated by machine learning approach is 0.216, Improved GA is 0.293, Deep Learning is 0.306, and proposed WCWWO+HAN is 0.354. The analysis using fitness is depicted in figure 10c). For the test case size 200, the fitness value obtained by the Machine learning, Improved GA, Deep learning, and proposed WCWWO+HAN are 0.368, 0.436, 0.468, and 0.483.

4.6 Comparative Discussion

This portion portrays the comparable discussion of various techniques using the interpreters, such as Fibonacci code, Fankuch-redux code, and spectral norm code. Table 1 presents a comparable discussion of developed WCWWO+HAN in correlating with current approaches, like Machine learning, Improved GA, and Deep learning by varying the test case size as 400. Considering test case as 400, MS value obtained by machine learning, Improved GA, Deep learning, and proposed WCWWO+HAN is 0.523, 0.551, 0.575, and 0.585. When the test case size is 400, MRR value obtained by developed WCWWO+HAN is 0.397, where MRR value obtained by current approaches, like machine learning is 0.283, Improved GA is 0.342, and Deep learning is 0.386. The fitness value

Figure 9. Analysis of the proposed WCWWO+HAN using Spectral norm code a) MS, b) MRR, and c) Fitness

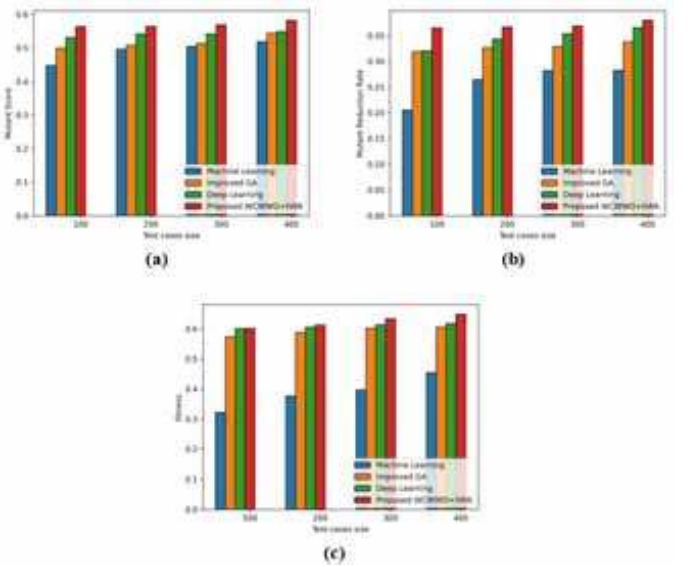
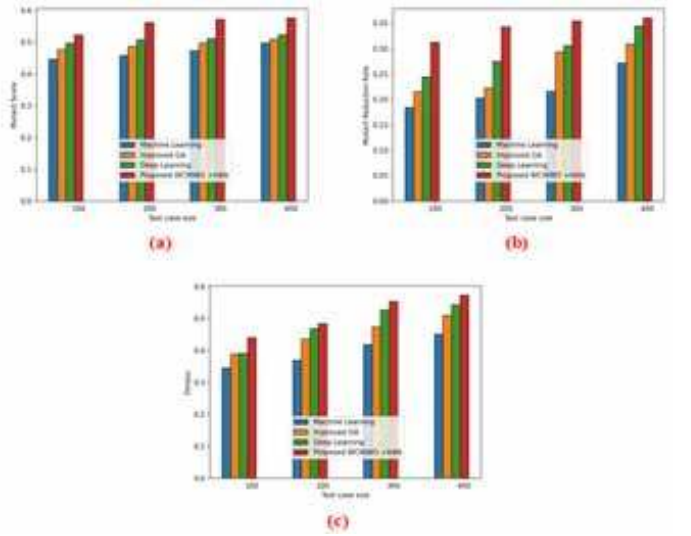


Figure 10. Analysis of the proposed WCWWO+HAN using meteor-contest benchmark a) MS, b) MRR, and c) Fitness



achieved by the machine learning approach is 0.520, Improved GA is 0.614, Deep learning is 0.626, and the proposed WCWWO+HAN is 0.652 for the test case size 400. From a table, it is distinctly displayed that the proposed WCWWO+HAN attained a maximum MS of 0.585, maximal fitness of 0.652 using Fibonacci code, and higher MRR 0.397 using the Fankuch-redux code. By considering the meteor-contest benchmark, the proposed WCWWO+HAN has the MS of 0.576, MRR of 0.360, and Fitness of 0.573.

Table 1.
Comparative Discussion

Interpreters	Metrics	Machine Learning	Improved GA	Deep learning	Proposed WCWWO+HAN
Fibonacci code	***MS***	0.523	0.551	0.575	**0.585**
	MRR	0.342	0.348	0.366	**0.375**
	Fitness	0.520	0.614	0.626	**0.652**
Fankuch-redux code	***MS***	0.530	0.555	0.561	**0.577**
	MRR	0.283	0.342	0.386	**0.397**
	Fitness	0.445	0.600	0.639	**0.641**
Spectral norm code	***MS***	0.518	0.543	0.548	**0.581**
	MRR	0.282	0.337	0.365	**0.379**
	Fitness	0.453	0.604	0.617	**0.648**
meteor-contest benchmark	***MS***	0.497	0.509	0.522	**0.576**
	MRR	0.272	0.309	0.345	**0.360**
	Fitness	0.450	0.509	0.543	**0.573**

5. CONCLUSION

This paper presents a robust approach for mutation testing employing multi-objective enabled optimization technique. Here, a generation of optimal suites is carried out using the proposed WCWWO where the mutation testing is integrated. The phases involved in the optimal test suite generation based on mutation testing are sample test program input, test suite generation, and optimization with mutation testing. The best test suite selection is decided using mutation testing where the test programs and the test suite to be optimized are presented to the HAN in order to find the MS. Besides, the mutant generator generates the mutant programs in such a way that the equivalent mutants are classified by HAN. Moreover, WCWWO is applied to generate and detect the finest test suite set, which satisfies multi-objectives, namely execution time, test suite size, MRR, and MS. The proposed WCWWO+HAN achieved enhanced performance with the maximal MS of 0.585, higher MRR of 0.397, and maximal fitness of 0.652, respectively. The proposed method is to classify the mutants into killed and survived mutants for each test suite. Also, it is useful to evaluate software quality. The future work would be the concern of developing advanced tools for handling the mutant generation and execution and also for enhancing the performance based on mutation testing.

REFERENCES

Aranega, V., Mottu, J. M., Etien, A., & Dekeyser, J. L. (2010). Traceability for mutation analysis in model transformation. *International Conference on Model Driven Engineering Languages and Systems*, 259-273.

Bashir, M. B., & Nadeem, A. (2017). Improved genetic algorithm to reduce mutation testing cost. *IEEE Access: Practical Innovations, Open Solutions*, *5*, 3657–3674.

Chen, L., & Zhang, L. (2018). Speeding up mutation testing via regression test selection: An extensive study. *IEEE 11th International Conference on Software Testing, Verification and Validation (ICST)*, 58-69.

Chen, T.Y., & Lau, M.F. (1998). A new heuristic for test suite reduction. *Information and Software Technology*, *40*(5), 347-354.

Clegg, B. S., Rojas, J. M., & Fraser, G. (2017). Teaching software testing concepts using a mutation testing game. *IEEE/ACM 39th International Conference on Software Engineering: Software Engineering Education and Training Track (ICSE-SEET)*, 33-36.

Dehmer, M., Chen, Z., Shi, Y., Zhang, Y., Tripathi, S., Ghorbani, M., Mowshowitz, A., & Emmert-Streib, F. (2019). On efficient network similarity measures. *Applied Mathematics and Computation*, *362*, 124521.

Dineva, K., & Atanasova, T. (2022). Cloud Data-Driven Intelligent Monitoring System for Interactive Smart Farming. *Sensors (Basel)*, *22*(17), 1–26.

Eskandar, H., Sadollah, A., Bahreininejad, A., & Hamdi, M. (2012). Water cycle algorithm–A novel metaheuristic optimization method for solving constrained engineering optimization problems. *Computers & Structures*, *110*, 151–166.

Fankuch-redux Code. (n.d.). https://pybenchmarks.org/u64q/benchmark.php?test=fannkuchredux&lang=python3&id=3&data=u64q

Fibonacci Code. (n.d.). https://pybenchmarks.org/u64q/benchmark.php? accessedtest=fibonacci&lang=python3&id=3&data=u64q

Fraser, G., & Arcuri, A. (2015). Achieving scalable mutation-based generation of whole test suites. *Empirical Software Engineering*, *20*(3), 783–812.

Gómez-Abajo, P., Guerra, E., de Lara, J., & Merayo, M. G. (2020). Wodel-Test: A model-based framework for language-independent mutation testing. *Software & Systems Modeling*, 1–27.

Inozemtseva, L., & Holmes, R. (2014). Coverage is not strongly correlated with test suite effectiveness. *Proceedings of the 36th international conference on software engineering*, 435-445.

Jabbarvand, R., & Malek, S. (2017). μDroid: An energy-aware mutation testing framework for Android. *Proceedings of the 11th Joint Meeting on Foundations of Software Engineering*, 208-219.

Jamil, M. A., Arif, M., Abubakar, N. S. A., & Ahmad, A. (2016). Software testing techniques: A literature review. *IEEE 6th International Conference on Information and Communication Technology for The Muslim World (ICT4M)*, 177-182.

Jatana, N., & Suri, B. (2020a). Particle Swarm and Genetic Algorithm applied to mutation testing for test data generation: A comparative evaluation. *Journal of King Saud University - Computer and Information Sciences*, *32*(4), 514-521.

Jatana, N., & Suri, B. (2020b). An improved crow search algorithm for test data generation using search-based mutation testing. *Neural Processing Letters*, *52*(1), 767-784.

Jia, Y., & Harman, M. (2010). An analysis and survey of the development of mutation testing. *IEEE Transactions on Software Engineering*, *37*(5), 649–678.

Jovanović, I. (2006). Software testing methods and techniques. *The IPSI BgD Transactions on Internet Research*, *5*(1).

Kasurinen, J. (2010). Elaborating software test processes and strategies. *IEEE Third International Conference on Software Testing, Verification and Validation*, 355-358.

Kirmani, M., & Wahid, A. (2015). Revised Use Case Point (Re-UCP) Model for Software Effort Estimation. *International Journal of Advanced Computer Science and Applications*, *6*(3), 65–71.

Kulkarni, S. B. (2022). Hybrid GOA and GA algorithm based Deep Belief Network for Network Controlled Vertical Handoff. *Journal of Networking and Communication Systems*, *5*(1).

Li, H., Wang, T., Zhang, M., Zhu, A., Shan, G., & Snoussi, H. (2019). Hierarchical Attention Networks for Image Classification of Remote Sensing Images Based on Visual Q&A Methods. *2019 Chinese Automation Congress (CAC)*, 4712-4717.

Lou, Y., Hao, D., & Zhang, L. (2015). Mutation-based test-case prioritization in software evolution. *IEEE 26th International Symposium on Software Reliability Engineering (ISSRE)*, 46-57.

Ma, L., Zhang, F., Sun, J., Xue, M., Li, B., Juefei-Xu, F., Xie, C., Li, L., Liu, Y., Zhao, J., & Wang, Y. (2018). Deepmutation: Mutation testing of deep learning systems. *IEEE 29th International Symposium on Software Reliability Engineering (ISSRE)*, 100-111.

Mao, D., Chen, L., & Zhang, L. (2019). An extensive study on cross-project predictive mutation testing. *12th IEEE Conference on Software Testing, Validation and Verification (ICST)*, 160-171.

Moon, S., Kim, Y., Kim, M., & Yoo, S. (2014). Ask the mutants: Mutating faulty programs for fault localization. *IEEE Seventh International Conference on Software Testing, Verification and Validation*, 153-162.

Naeem, M. R., Lin, T., Naeem, H., & Liu, H. (2020). A machine learning approach for classification of equivalent mutants. *Journal of Software: Evolution and Process*, *32*(5), e2238.

Naeem, M. R., Lin, T., Naeem, H., Ullah, F., & Saeed, S. (2019). Scalable mutation testing using predictive analysis of deep learning model. *IEEE Access: Practical Innovations, Open Solutions*, *7*, 158264–158283.

Papadakis, M., Kintis, M., Zhang, J., Jia, Y., Le Traon, Y., & Harman, M. (2019). Mutation testing advances: An analysis and survey. *Advances in Computers*, *112*, 275–378.

Papadakis, M., Kintis, M., Zhang, J., Jia, Y., Traon, Y. L., & Harman, M. (2019). Chapter Six - Mutation Testing Advances: An Analysis and Survey. *Advances in Computers*, *112*, 275–378.

Python Interpreters Benchmarks. (n.d.). https://pybenchmarks.org/

Python Interpreters Benchmarks Dataset. (n.d.). Available at https://pybenchmarks.org/u64q/benchmark.php?test=meteor&lang=pypy3&id=3&data=u64q

Rani, S., Dhawan, H., Nagpal, G., & Suri, B. (2019). Implementing Time-Bounded Automatic Test Data Generation Approach Based on Search-Based Mutation Testing. Progress in Advanced Computing and Intelligent Engineering, 113-122.

Singh, P. K., Sangwan, O. P., & Sharma, A. (2013). A systematic review on fault based mutation testing techniques and tools for Aspect-J programs. *3rd IEEE International Advance Computing Conference (IACC)*, 1455-1461.

Spectral Norm Code. (n.d.). https://pybenchmarks.org/u64q/benchmark.php?test=spectralnorm&lang=python3&id=3&data=u64q

Tiwari, M., Bansal, V., & Bajaj, A. (2012). Ant Colony Optimization: Algorithms of Mutation Testing. *International Journal of Engineering Research & Technology (Ahmedabad)*, *1*(9), 1–8.

Wang, J., Dong, G., Sun, J., Wang, X., & Zhang, P. (2019).Adversarial Sample Detection for Deep Neural Network through Model Mutation Testing. *Proceeding of IEEE/ACM 41st International Conference on Software Engineering (ICSE)*.

Zheng, Y. J. (2015). Water wave optimization: A new nature-inspired metaheuristic. *Computers & Operations Research*, *55*, 1–11.