I.J. Modern Education and Computer Science, **2022, 4, 43-56**
Published Online August 2022 in MECS (http://www.mecs-press.org/)
DOI: 10.5815/ijmecs.2022.04.04

Hybridization of Buffalo and Truncative Cyclic Gene Deep Neural Network-based Test Suite Optimization for Software Testing

T. Ramasundaram
Research Scholar, Department of Computer Science, Periyar University, Salem,Tamilnadu, India
Email: profram01@yahoo.in

V. Sangeetha
Assistant Professor and Head, Department of Computer Science, Government Arts and Science College, Pappireddipatti, Tamil Nadu, India
Email: sangee759@gmail.com

Received: 21 November 2020; Accepted: 19 January 2021; Published: 08 August 2022

Abstract: Software testing is the significant part of the software development process to guarantee software quality with testing a program for discovering the software bugs. But, the software testing has a long execution time by using huge number of test suites in the software development process. In order to overcome the issue, a novel technique called Hybridized Buffalo and Truncation Cyclic Gene Optimization-based Densely Connected Deep Neural Network (HBTCGO-DCDNN) introduced to improve the software testing accuracy with minimal time consumption. At first, the numbers of test cases are given to the input layer of the deep neural network layer. In the first hidden layer, the test suite generation process is carried out by applying the improved buffalo optimization technique with different objective functions namely time and cost. The improved buffalo optimization selects optimal test cases and generates the test suites. After the generation, the redundant test cases from the test suite are eliminated in the reduction process in the second hidden layer. The Truncative Cyclic Uniformed Gene Optimization technique is applied for the test suite reduction process based on thefault coverage rate. Finally, the reduced test suites are obtained at the output layer of the deep neural network The experimental evaluation of the HBTCGO-DCDNN and existing methods are discussed using the test suite generation time, test suite reduction rate as well as fault coverage rate. The comparative results of proposed HBTCGO-DCDNN technique provide lesser the generation time by 48% and higher test suit reduction rate by 19% as well as fault coverage rate 18% than the other well-known methods.

Index Terms: Software testing, Densely Connected Deep feedforward Neural Network, test suites generation, improved buffalo optimization, testsuites reduction, Truncative Cyclic Uniformed Gene Optimization.

1. Introduction

Software testing is the exclusive performance in the software improvement lifecycle to guarantee the quality of the software program. One of the significant focuses in the software testing process is the cost and time. Test suites are the set of test cases to test a software program. An optimal test suite generation is a significant process that can cover the entire fault with minimum time. The selected test suites need to satisfy the testing requirements and revealing faults. Several optimization techniques have been developed for finding optimal test suites from exhaustive test suites.

Annibale Panichella et al. introduced a Dynamic Many-Objective Sorting Algorithm (DynaMOSA) in [1] for generating the test case to perform the coverage testing. The designed algorithm considerably covers more faults but the optimal test suite selection was not efficiently performed. In [2], Arun Prakash Agrawal et al. presented a fault coverage-based test suite optimization (FCBTSO) framework for minimizing the number of test suites. The designed technique minimizes the execution time to find the optimized test suite but the deep analysis was not performed.

Testing is a vital role to provide software quality in software. The optimal test suite generation to discover a possible fault and thereby enhances the software quality. Test case generation is multi objective problem is discussed in [1, 2] and it is an open research problem. Many research works have been designed for test suite generation and optimal test suites generation in software. In addition, enhancing the software quality within the software program application is a major challenging concern. However, the optimal test suite selection was not executed. But, it failed to perform deep analysis. Then, the main limitation of conventional method such as more time and cost are utilized for optimal test suite

generation. In order to address the above problem, HBTCGO-DCDNN is introduced. The major objective of this paper is to select optimal test suites using a HBTCGO-DCDNN and reduce the time and cost to handle the existing problem. The proposed HBTCGO-DCDNN technique is developed with the implementation of improved buffalo optimization, truncative cyclic uniformed gene optimization to achieve better performance of maximum reduction rate, coverage rate, and minimum time consumption than the state of art methods.

The primary contribution of the HBTCGO-DCDNN is summarized as follows,

1) This work presents a new heuristic approach called HBTCGO-DCDNN for Fault Coverage-Based Test Suite generation and optimization with minimum time.
2) Proposed HBTCGO-DCDNN technique adopted by Densely Connected Deep Neural Network with the buffalo optimization. The buffalo optimization technique in the hidden layer for learning the test suites based on time and cost. The test cases which having lesser execution time and cost are selected for optimal one and generates the test suites.
3) Proposed HBTCGO-DCDNN technique uses the Uniformed Gene Optimization technique to have an ability to identify and select the best optimal test cases for software testing. The proposed gene optimization chooses an optimal test suite based on fault coverage rate.
4) An extensive evaluation of HBTCGO-DCDNN is performed with three baseline approaches for comparison. The technique is tested based on the various performance metrics.

This paper is organized as follows. Existing recent research works are described in section II. The proposed HBTCGO-DCDNN with generation and optimization of test suites are described in section III. Section IV shows experimental settings with the Web chess dataset and also provides the parameter description. In section V, comparative experimental results are discussed for the proposed and existing software test case based optimization techniques. Finally, the conclusion is made in section VI.

2. Related Works

In [3], Alessandro Marchetto at al. developed a Multi-Objective test suites REduction (MORE+): a three-dimension approach to minimize the number of test suites. But the designed approach was not efficient for minimizing the test suites based on cost-effective approaches. In [4], Abdullah B. Nasser at al. designed a flower pollination algorithm (FPA) for generating the test suites. The designed algorithm fails to provide optimum generating results.

In [5], Neha Gupta et al. introduced a mutant coverage-based multi-objective method to generate an optimal test suite having the ability to identify and locating the faults from the software program. But the time consumption of the optimal test suite generation was high. Rosziati Ibrahim et al. developed a Eclipse Plug-in Tool in [6] to optimize the creation of test cases from source code with lesser time. But the test case reduction process was not performed.

In [7], Divya Taneja et al. presented a Linear Regression technique for the minimization of test cases. The deigned technique efficiently decreases the execution time for reducing the test cases. Dharmveer Kumar Yadav and Sandip Dutta examined a Regression-based test suite selection in [8] to test the software programs. The designed approach was used to choose the best feasible test cases but the time consumption was not minimized.

In [9], AutO. ÖrsanÖzener and Hasan Sözer discussed the multi-criteria test suite minimization depend on linear programming. The designed technique failed to implement the evaluation with larger datasets, and also failed to compare the results with the heuristics or greedy algorithms to increase the effectiveness and efficiency of test suite minimization. In [10], Handing Wang et al. introduced multi-fidelity optimization to develop the test suites. Though the designed optimization algorithm reduces the computational cost, the higher fault coverage rate was not obtained.

Wei Zheng et al. designed a mutation score (MS)-guided optimization techniques in [11] for the test suite reduction process with minimum cost and code coverage criteria. But the deep learning process was not implemented to obtain a better solution. In [12], Akram Kalaee and Vahid Rafe developed a various search-based optimization techniques to generate test suites for model-based software testing. But it failed to apply the Machine Learning techniques for the testing problem to increase the effectiveness of the test suites generation.

Yoo-Min Choi and Dong-Jin Lim introduced a test-case generation method in [13] to decrease the number of test suites for fault localization. The method reduces the execution time but the accurate fault coverage rate was not obtained. In [14], Abha Maru et al. designed a backpropagation neural network for identifying the accurate location of the fault from the given software program. But the designed network failed to detect the many faults accurately.

In [15], Andrea Arcuri designed a Many Independent Objective (MIO) algorithm to create the test suites. The designed algorithm achieved better performance but it failed to consider the multiple objectives. Xiaoan Bao et al. described an improved adaptive genetic algorithm (IAGA) in [16] to generate the test cases by preserving population diversity. Haifeng Wang et al. designed an information entropy-based test case reduction (IETCR) scheme in [17] for fault localization. The designed scheme increases the reduction rate but accurate execution time minimization was not achieved.

Pedro Delgado-Pérez and Inmaculada Medina-Bulo designed an Evolutionary Mutation Testing (EMT) in [18] for reducing the number of test suites. But the selected test suites were not efficient to cover the more faults. In [19], Warda Elkholy et al. designed a model checking intelligent avionics system for test case generation using multi-agent systems. The designed system failed to express the quantitative coverage measure for the generated test cases. Sonali Pradhan et al. introduced a set of algorithms in [20] to produce test cases from a statechart diagram depends on different coverage criteria. But the designed algorithm was not performed the cost-effectiveness analysis for test case generation. R. Anbunathan and Anirban Basu investigated a Genetic Algorithm (GA) in [21] for creating the Basis Path (BP) tests cases by node coverage. But, the generation time was not higher. In [22] Abhinandan H. Patil et al. developed a Residual Test Coverage Algorithm and Statistical Techniques for regression test suite prioritization. However, the test suite reduction rate was minimized. Abhinandan H. Patil et al. presented Statistical techniques in [23] to regression test suites. But, the coverage rate was not enhanced.

3. Methodology

A novel technique called HBTCGO-DCDNN is introduced for achieving better software quality testing. The testing of software is made to find the defects. If a fault exists, then it requires be detecting and then resolving to increase the reliability of software programs. The proposed fault detection system helps in detecting the faults accurately and the technique produces minimized test suites having the capability of detecting all detected faults from the given software programs. The research objective is to achieve better software quality testing with lesser time. The innovation of improved buffalo optimization and truncative stochastic cyclic uniformed gene optimization is used in the proposed HBTCGO-DCDNN technique to increase the software quality. In the present work, a proposed technique uses test suit generation and test suite minimization together for fault detection. The basic work flow of the proposed HBTCGO-DCDNN is shown in the architectural diagram.

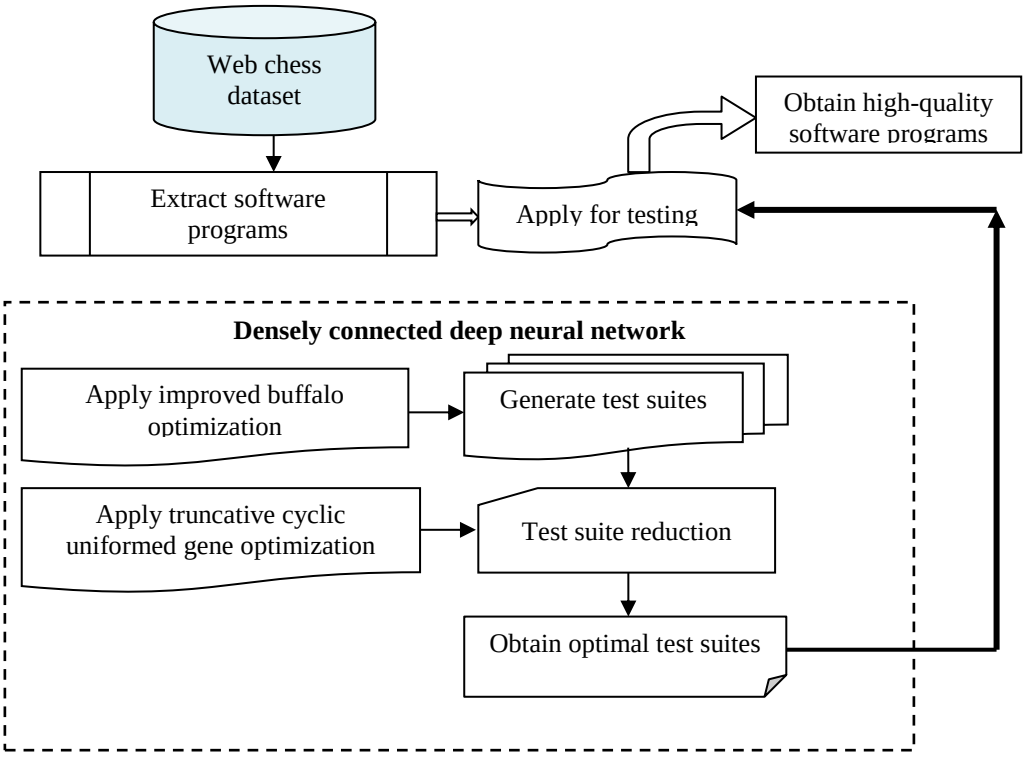

Fig. 1. Basic workflow of proposed HBTCGO-DCDNN technique

Fig.1 demonstrates the basic architecture of the proposed HBTCGO-DCDNN. Let us consider the number of software programs$S_i \in (s_1, s_2, s_3, \ldots s_n)$' taken from the web chess dataset. To test the software codes taken from the web chess dataset, the optimal test suites are generated. The proposed HBTCGO-DCDNN uses the densely connected deep forward neural network for learning the given input with numerous layers connected between them. The layers include neurons like nodes that are connected from one layer to the next layer with regulating weights. The densely connected deep network structure uses one input, output, and two hidden layers for deeply analyzing the given input.

In Fig. 2, the structural design of a densely connected deep forward neural network is presented. Fig. 2 consists of the input layer which receives the input as test cases$\tau_i \in \tau_1, \tau_2, \ldots . \tau_n$.

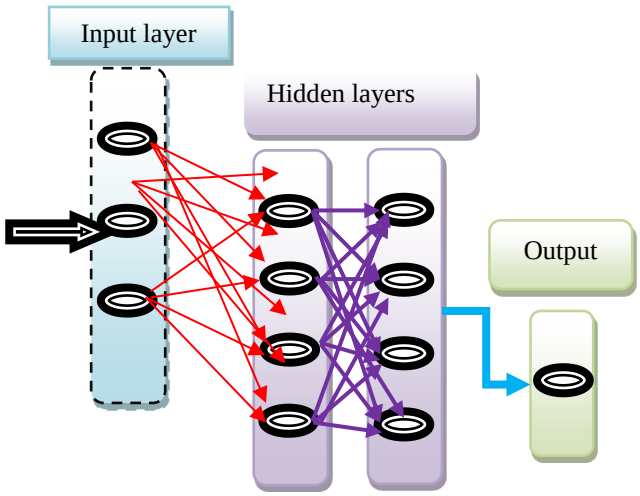

Fig. 2. Structural design of densely connected deep forward neural network

These test cases are formed to construct the test suites. The neuron activity at the input layer at a time‘t’ is expressed as follows,

$$A(t) = c + \sum_{i=1}^{n} \beta * \tau_i(t) \quad (1)$$

From (1), $A(t)$ indicates an input layer, ‘$\tau_i(t)$’ indicates an input i.e. test cases, ‘β’ indicates a weight to strengthen the connections, ‘c’ indicates a bias and it simply stores the value of 1. Then the input is forwarded into the hidden layer for learning the input for generating the test suites.

In a densely connected deep forward neural network, a hidden layer is located between the input and output layer of the structural design, in which the test suit generation process is performed. To generate the test suites, the optimal test cases are identified using the improved buffalo optimization algorithm. On the contrary to existing optimization, the algorithm considers the multiple objective functions in the test suite generation. Hence the name is called improved buffalo optimization. Buffalo optimization is inspired by the behavior of buffalos in theAfrican forests. The developed optimization is a meta-heuristic algorithm to detect the position of the best buffalo among the population. The designed optimization is a robust and efficient algorithm than the other optimization algorithm since it guarantees a fast convergence rate with the utilization ofless learning parameters. The optimization technique starts to perform the learning process in the first hidden layer with the population's initialization. The population of the buffalo is randomly initialized. Here, the Buffalo’s are related to the test cases. Therefore, the initialization process is expressed as follows,

$$\tau_i = \tau_1, \tau_2, \ldots . \tau_n \quad (2)$$

After the initialization of the Buffalo’s in search space, the fitness is computed based on the time and cost. The time is the amount of time to test the set of test cases based on the time consumed per person to run the test cases. It is evaluated as given below,

$$t_{T_i} = S * TRPP \quad (3)$$

Where,t_{T_i} denotes a time, ‘S’ indicates the size of test cases, ‘$TRPP$’ represents the time required per person (i.e. hour). Another objective function is a cost that is executing the set of test cases based on the effort and cost per man for a day. The cost of the test case ‘c_{T_i}’ is evaluated as follows,

$$c_{T_i} = Effort * CPMD \quad (4)$$

Where c_{T_i} indicates a cost, ‘$CPMD$’ represents the cost per man-day which is evaluated in terms of a dollar($). Then the fitness is estimated based on the above said time and cost estimation.

$$f(x) = \arg\min\{t_{\tau_i}, c_{\tau_i}\} \quad (5)$$

Where $f(x)$ points out the fitness of each test case, $\arg min$ indicates an argument of the minimum function. Therefore, the test cases are generated based on the minimum time and cost. Based on the fitness value, the position of each individual gets updated as follows,

$$Q\ (t+1)\ = Q(t) + r_1[f(x)_b - E_k] + r_2(Q_b - E_k) \tag{6}$$

From (6), $Q\ (t+1)$ indicates an updated buffalos' exploitation of the 'k^{th}' buffalo, $Q(t)$ indicates the current position of the buffalos', r_1 and r_2 denotes a learning parameter and it ranges from 0.1 to 0.6, $f(x)_b$ specifies the best fitness of buffalo's, Q_b indicates an individual buffalo's best location, E_k indicates the current exploration value of the buffalos. Then the updated value of the exploration is determined as follows,

$$E_k(t+1) = \frac{[E_k + Q(t)]}{R} \tag{7}$$

Where $E_k(t+1)$ denotes an updated value of the exploration, 'R' denotes the movement of buffalo and usually is set to 1. The above-said process is repeated until the convergence is met. At last, the optima test cases are obtained with lesser time to generate the test suites for testing the quality of the software program. The flow process of the algorithm is expressed as follows,

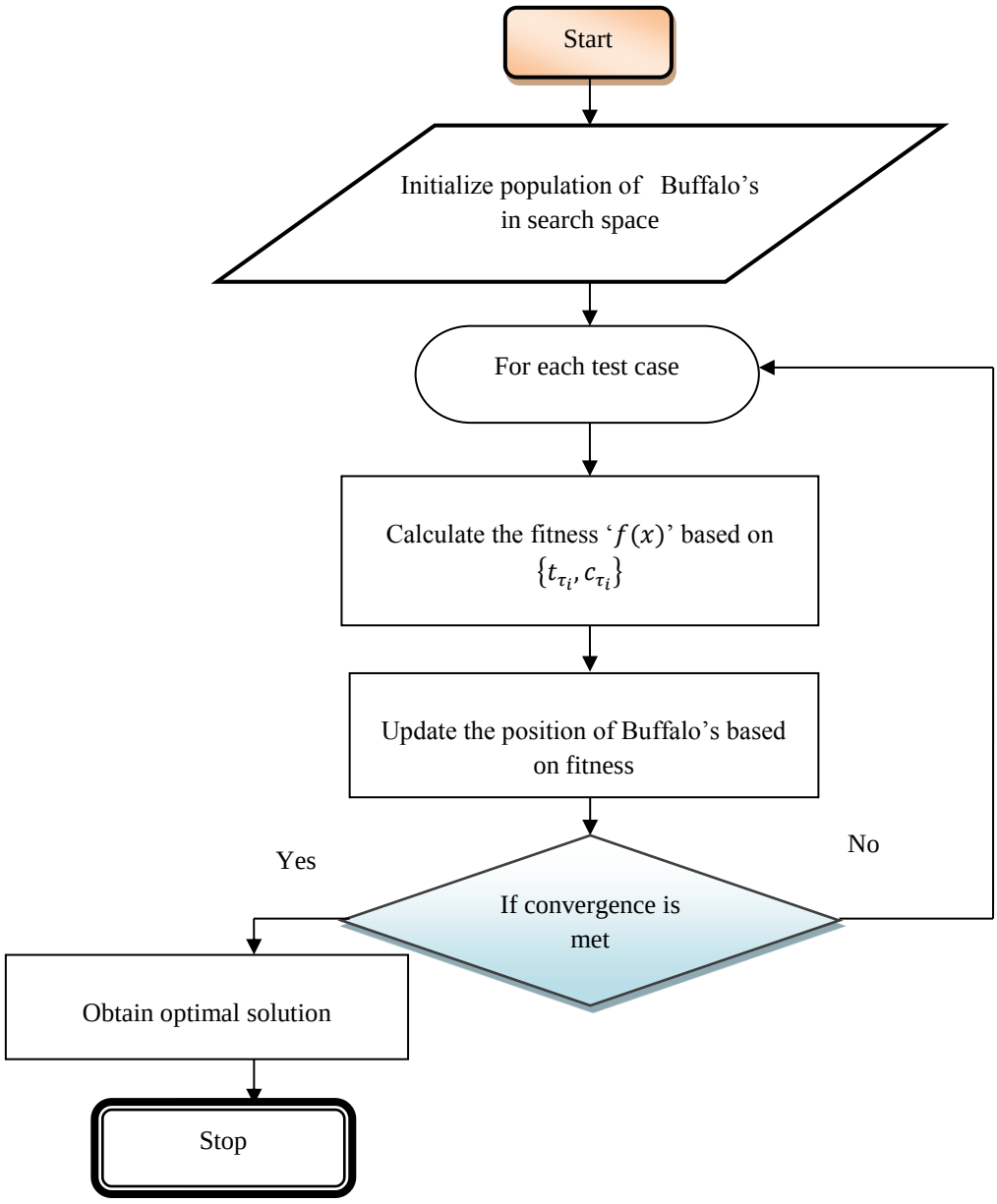

Fig. 3. Flow process improved buffalo optimization-based test suit generation

Fig. 3 demonstrates the flow process of improved buffalo optimization-based test suit generation to find the test cases. Then the generated test suites are transferred into the next hidden layer.

A. Second Hidden Layer: Truncative Cyclic Uniformed Gene Optimization

In the second hidden layer, truncative cyclic uniformed gene optimization is performed to reduce the test cases for testing the quality of the software program. The proposed gene optimization technique receives the input as generated test suites. The flow process of Truncative cyclic uniformed gene test suite optimization is illustrated in Fig. 4.

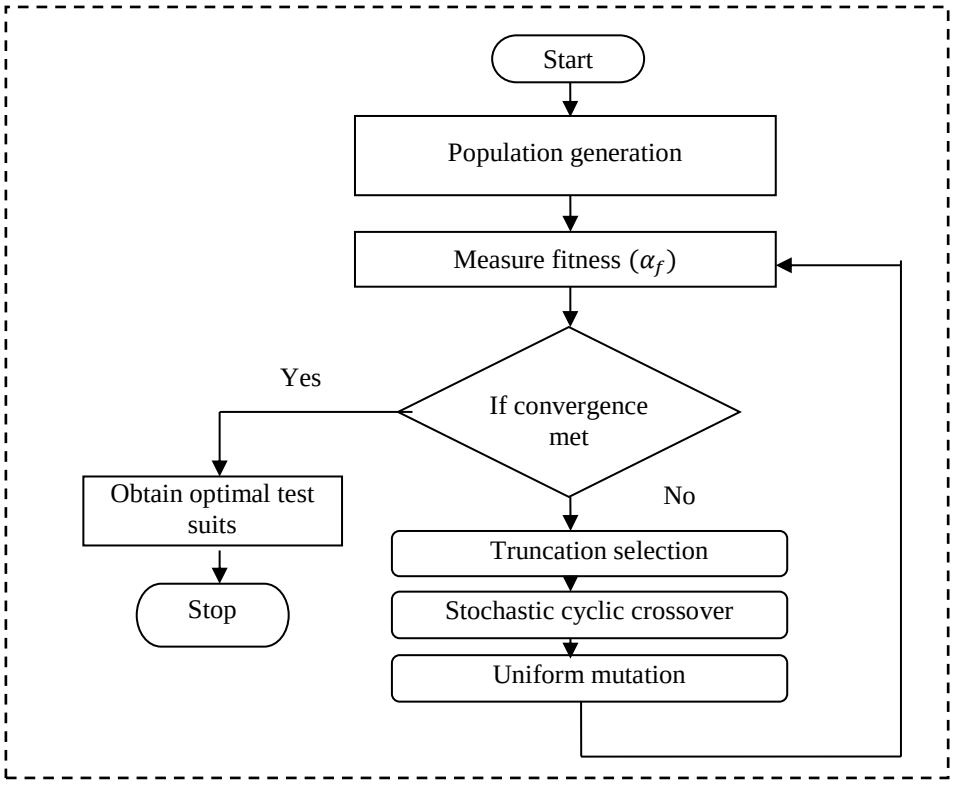

Fig. 4. Flow process of Truncative cyclic uniformed gene optimization

The proposed technique starts to generate the population of the generated test suites as follows,

$$`TS_i = TS_1, TS_2, .., TS_N \tag{8}$$

Where TS_i indicates the 'N' number of generated test suites. For each generated test suite, the fitness is estimated based on the fault coverage rate as given below,

$$\alpha_f = FCR = \frac{NF_{TS}}{T_{NF}} \tag{9}$$

Where 'α_f' denotes the fitness of the test suite,FCR denotes a fault coverage rate, NF_{TS} denotes a higher number of faults covered by a test suite, T_{NF} indicates a total number of faults in a given software program application.If the algorithm did not meet the convergence, then the bio-inspired genetic operators such as truncation selection, cyclic crossover, and uniform mutation are carried out to find the optima test suites. These three genetic operators are explained as given below,

B. Truncative Selection

The selection is the first operator of the gene optimization for choosing the parent chromosomes (i.e. test suites) from the initial population. The proposed uses the truncative selection for choosing the current best quantum chromosomes based on their current fitness. Initially, test suites are sorted in a descending order based on their current fitness (F). After sorting the test suites, the truncation threshold is set to choose the best individuals (i.e. Chromosomes). Only the fitness above the threshold is chosen as parents for the next recombination process to generate new offspring.

$$TS_s = \begin{cases} \alpha_f > \omega_{\alpha_f};\ selectedasparentchromosomes \\ otherwise;\ \ notselectedasparentchromosomes \end{cases} \quad (10)$$

WhereTS_s indicates a truncative selection of parent chromosomes, 'α_f' represents a fitness of the chromosomes, ω_{α_f} symbolizes the truncation threshold. In this way, the best individuals are chosen as parents for the recombination process.

C. Stochastic Cyclic Crossover

Crossover is the genetic operator to combine the two-parent chromosomes forcreating a new offspring. It is one system to stochastically (i.e. having a random probability) generate new offspring's from an existing population.

The Stochastic cyclic crossover generates one offspring's from the two parents. Let us consider a two-parent chromosomes 'p ,$q$randomly in the forms of binary representation as follows,

'p Binary representation is b_1, b_2, b_3, b_4, b_5
q Binary representation is c_1, c_2, c_3, c_4, c_5

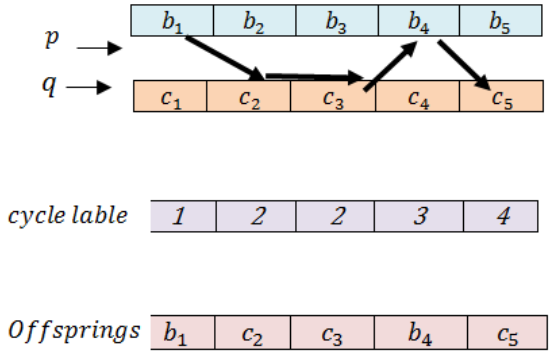

Fig. 5. Stochastic cyclic crossover based offspring generation

Fig. 5 illustrates the process of Stochastic cyclic crossover is described to generate offspring from parent chromosomes. The cyclic crossover is performed based on the combination of two well-defined parent chromosomes that provide the newly adopted ones. By applying the crossover process, the cycle label is assigned as even and odd numbers. If the cycle label is an odd number, the offspring are generated from the parent 'p'. If the cycle label is an even number, the offspring are generated from the parent 'q'. In this way, the offspring are generated from the parents. The generated string length of the offspring is equal to the total string length of their parents.

D. Uniform Mutation

After generating an offspring, the uniform mutation is carried out to preserve the genetic diversity from one generation to the next generation. Uniform Mutation is the process of replacing the selected chromosome values from the offspring generated by uniform mutation. As a result of mutation, a new offspring is generated as follows. Let us consider the offspring generated from the stochastic cyclic crossover is considered as follows,

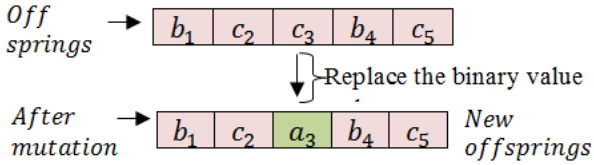

Fig. 6. Uniform mutation

Fig. 6 illustrates the uniform mutation for obtaining the offspring. This process is continued until the convergence is met in a population and discoversthe most optimal number of test suites.The higher-fitness offspring reach the optimal solution in the search space. It determines the minimum number of test suites to improve software quality. Then the output of the second hidden layer is expressed as follows,

$$B(t) = [\sum_{i=1}^{n} \beta * \tau_i(t)] + [\beta_1 * w_{t-1}] \quad (11)$$

Where,$B(t)$indicates an output of hidden layer at a time instance 't', β denotes a weight of input and hidden, $\tau_i(t)$ indicates an input (i.e. test cases), β_1 represents the weight of hidden layers, w_{t-1} indicates an output of the first hidden layer. The obtained results are fed into the output visible layer for obtaining the final optimized results,

$$C(t) = [\beta_3 * B(t)] \quad (12)$$

Where'$C(t)$ specifies the output of the densely connected deep forward neural network at the time 't', β_3 designates a weight between the hidden and output layer, $B(t)$ indicates an output of the hidden layer. As a result, an optimal number of test suites are obtained at the output layer for testing the given software program taken from the webchess dataset. The algorithmic process of the proposed technique is explained as follows,

// Algorithm 1: Hybridized Buffalo and Truncative Cyclic Gene Optimization-based Densely Connected Deep Neural Network

Input: Webchess, set of test cases '$\tau_i = \tau_1, \tau_2, .. \tau_n$

Output: Optimized number of Test Suites

Begin

Collect the number of test cases '$\tau_i = \tau_1, \tau_2, .. \tau_n$ in the input layer

\\ **hidden layer 1**

1. Randomly initialize a population of ' $\tau_i = \tau_1, \tau_2, .. \tau_n$'
2. Initialize learning parameters r_1 and r_2
3. **For** each τ_i
4. Calculate time and cost
5. Compute fitness '$f(x)$'
6. **While** (t <Max_iter)
7. **if** $\left(f(x_i) < f(x_j)\right)$ **then**
8. Update buffalos' exploitation $Q(t+1)$
9. Update the location of buffalos $E_k(t+1)$
10. **end if**
11. **t= t+1**
12. **end while**
13. **end for**
14. **Return the optimal test cases to generate test suites**

\\ **hidden layer 2**

15. Initialize the population of a number of test suite generated $TS_i = TS_1, TS_2, .., TS_N$
16. **For each** TS_i
17. Measure fitness value' α_f'
18. **If (*convergencemet*) then**
19. Select 'TS_i' as an optimal test suite
20. **Else**
21. **Select** TS_i based on the fitness
22. **G**enerate a new offspring's from parent chromosomes p, q
23. Perform uniformmutation by replacing the binary value and obtain new offspring
24. Go to step 18
25. Terminate the process until the specified condition is met
26. **end if**
27. **end for**
28. Obtain a minimal number of test suites at the output layer

End

Algorithm 1 describes step by step process of optimized test suite generation and minimization for improving the testing performance of software quality. The optimization-based deep learning process effectively performs both test suite generation and reduction. The number of test cases is collected and it is given to the input layer. In the first hidden layer, improved buffalo optimization is applied for generating the test suites by finding the time and cost-efficient test cases. Finally, the test suites are generated in the first hidden layer. In the second hidden layer, the test suit reduction process is carried out by applying the truncative stochastic cyclic uniformed gene optimization. The optimization technique measures the fault coverage rate of each generated test suites. The test suites which having the maximum fault coverage rate are taken as final test suites and other test suites are removed. In this way, the test suites reduction process is performed.

4. Experimental Evaluation

In this section, the proposed HBTCGO-DCDNN and existing methods DynaMOSA [1], FCBTSO [2], MORE+ [3] are implemented in Java Language using the Webchess dataset. In the Webchess dataset, the numbers of test cases are collected. The Webchess dataset is taken from the https://sourceforge.net/projects/webchess/. The dataset comprises the 28 PHP software programs used for testing. In the HBTCGO-DCDNN, the test suites are chosen and generated with aid of improved buffalo optimization technique. To minimize the software testing time and increases the testing quality, a less number of test suites are selected from the generated test suites based on fault coverage. In order to reduce the test cases, reduction process is applied. By using Truncative Cyclic Uniformed Gene Optimization technique, the optimal test cases based on time and cost is discovered. With the reduced test suites, the software testing process is carried out with the PHP software programs taken from the Webchess dataset and it helps to increases the testing quality performance and minimizes the testing time.

The implementation is conducted with the hardware specification of Windows 10 Operating system, core i3-4130 3.40GHZ Processor, 4GB RAM, 1TB (1000 GB) Hard disk, ASUSTek P5G41C-M Motherboard, Internet Protocol. For accomplishing the experimental evaluation, the HBTCGO-DCDNN considers a number of test suite in the range of 15 to 150 from the Webchess dataset.

A. Data Analysis

The performance of the proposed HBTCGO-DCDNN technique is evaluated with the existing methods in terms of test suite generation time, test suite reduction rate as well as fault coverage rate.

Test suite generation time: It is defined as the amount of time taken by the algorithm to generate the test suites. The formula for measuring the test suite generation time is expressed as given below,

$$TSGT = N * t\{OTSG\} \tag{13}$$

Where $TSGT$ denotes a test suite generation time, 'N' designates a total number of test suites, and '$t(OTSG)$' denotes a time consumed to generate one test suite. The overall test suite generation time is measured in terms of milliseconds (ms).

Test Suite Reduction Rate: It is defined as the ratio of a number of test suites that are correctly selected as optimal to the total number of test suites generated. Therefore, the test suite reduction rate is calculated using the following expression,

$$TSRR = \left(\frac{n_{OTS}}{N}\right) * 100 \tag{14}$$

Where $TSRR$ indicates a Test Suite Reduction Rate and 'N' denotes a total number of test suites and 'n_{OTS}' symbolizes a number of test suites correctly selected as optimal. The test suite reduction rate is measured in percentages (%).

Coverage rate: It is determined as the ratio of a number of test suites that cover the more faults in the given software program taken from the Webchess dataset. The coverage rate is mathematically estimated as follows,

$$CR = \left(\frac{n_{CF}}{N}\right) * 100 \tag{15}$$

Where CR denotes a coverage rate of test suites generation, 'N' stands for the total number of test suites, 'n_{CF}' refers to the number of test suites that cover the more faults in the software program. Therefore, the coverage rate is measured in percentage (%).

5. Performance Analysis

Performance analysis of HBTCGO-DCDNN and existing methods DynaMOSA [1], FCBTSO [2], MORE+ [3] are discussed in this section concerning different parameters such as test suite generation time, test suite reduction rate as well as coverage rate. The efficacy of HBTCGO-DCDNN and conventional state of the art methods are determined based on the tables and graphical representation.

Table 1. Test Suite Generation Time

Test suites (Numbers)	Test suite generation time (ms)			
	HBTCGO-DCDNN	DynaMOSA	FCBTSO	MORE+
15	7	16	12	14
30	9	24	15	21
45	12	27	18	25
60	13	30	21	27
75	15	34	27	30
90	17	38	30	35
105	20	39	32	36
120	21	42	34	40
135	23	43	35	41
150	25	50	38	47

Table 1 given above shows that the comparative analysis of the test suite generation time using four different techniques namely HBTCGO-DCDNN, DynaMOSA [1], FCBTSO [2], MORE+ [3]. Totally ten runs are performed for each technique with different counts of inputs. Among the four methods, the time consumption is said to be reduced using HBTCGO-DCDNN. In the first run, 10 test suites are generated through the time consumption of the HBTCGO-DCDNN is $7ms$ whereas the time consumption of the DynaMOSA [1], FCBTSO [2], MORE+ [3] are $16ms$, $12ms$, $14ms$ respectively with similar counts of input. The obtained results of the proposed technique are compared to the existing results. The average of ten results indicates that the HBTCGO-DCDNN consumes a lesser amount of time and the performance is increased by 54%, 39%, 50% when compared to existing [1] [2] [3] respectively.

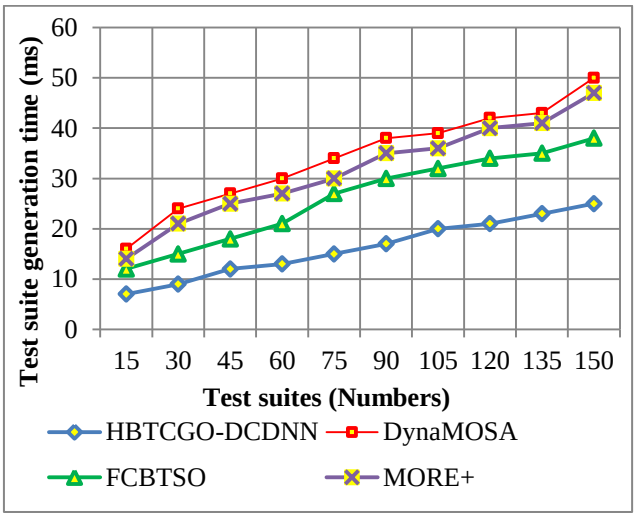

Fig. 7. Test suite Vs Test suite generation time

Fig. 7 given above indicate the experimental results of the test suite generation time using four different methods. The generated test suites are increased simultaneously the time consumption also gets increased for all the methods since the input counts get increased. Comparatively, Fig. 7 noticed that the HBTCGO-DCDNN achieves better performance in terms of decreases the test suites generation time. This significant development is achieved by applying the improved buffalo optimization with various objective functions based on time and cost function. The developed optimized technique accurately finds the optimal test case thus creates the test suites. This in turn obtains the optimal test suites with lesser time consumption.

Table 2.Test suite reduction rate

Test suites (Numbers)	Test suite reduction rate (%)			
	HBTCGO-DCDNN	DynaMOSA	FCBTSO	MORE+
15	93	67	80	73
30	93	73	87	80
45	98	78	89	82
60	95	77	88	85
75	97	73	85	83
90	96	76	82	80
105	95	78	86	81
120	96	73	83	80
135	95	80	89	85
150	94	79	87	83

Table 2 exhibits the performance analysis of the test suite reduction rate versus the number of test suites generated. The observed outcomes of a test suite reduction rate are reduced by using HBTCGO-DCDNN. Let us consider the 15 test suites and the 14 test suites are correctly selected as optimal and the reduction rate of the proposed technique is 93%. Whereas, by applying [1][2] [3], 10,12,11 test suites are correctly chosen from the 15 test suites considered for experimental evaluation. The obtained test suite reduction rate of DynaMOSA [1], FCBTSO [2], MORE+ [3] is 67%, 80% and 73% respectively. Likewise, nine remaining results are observed for each method with the different counts of inputs. Ten reduction rates are observed and the results are compared with existing methods. The comparison results show that the test suite reduction rate of considerably reduced by 27 % when compared to [1], 11% when compared to [2], 17% when compared to [3]. The graphical illustration of the test suite reduction rate is illustrated in Fig. 8.

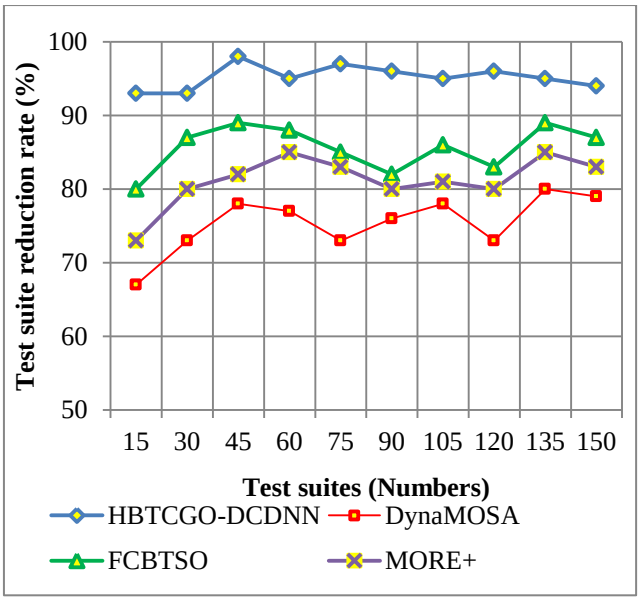

Fig. 8. Test suites Vs Test suite reduction rate

Fig. 8 depicts the analysis of the test suite reduction rate with different methods. The graphical plot indicates that the numbers of test suites are taken on the horizontal axis whereas the test suite reduction rates are observed at the vertical axis.On the basis of the observed results, it states that the test suites reduction rate is increased by applying HBTCGO-DCDNN that preserve their capability in detecting faults as compared with other related approaches. Among these approaches, it seems that HBTCGO-DCDNN uses the truncative cyclic uniformed gene optimization for finding the optimal test suites to test the software programs taken from the web chess dataset. The gene optimization finds the optimal test suites based on the fault coverage rates. The maximum fault coverage rates of the test suites are selected as optimal for testing the given software programs. The observed results suggest that HBTCGO-DCDNN achieves better results than the existing approaches.

Table 3.Coverage rate

Test suites (Numbers)	Coverage rate (%)			
	HBTCGO-DCDNN	DynaMOSA	FCBTSO	MORE+
15	93	60	73	67
30	90	70	80	77
45	89	69	82	73
60	92	77	87	80
75	93	76	85	81
90	94	78	87	82
105	92	75	86	79
120	94	80	88	83
135	92	81	89	84
150	95	80	87	83

Tables 3 show the results of a coverage rate versus the number of test suites with four different methods namely HBTCGO-DCDNN, DynaMOSA [1], FCBTSO [2], MORE+ [3]. From the table, the first column refers to the number of test suites are presented. The other columns indicate the results obtained by the coverage rate of different techniques. The obtained values noticed that the HBTCGO-DCDNN achieves a higher coverage rate when compared to other related approaches. Let us consider 15 test suites, the 14 test suites are that covers the more faults in the software program and the coverage rate of HBTCGO-DCDNN is 93% and the coverage rate of the other two existing methods are 60%, 73%,67% using DynaMOSA [1], FCBTSO [2], MORE+ [3]. The average of ten-value of the coverage rate is increased by 25%, 10%, and 18% using HBTCGO-DCDNN when compared to existing methods.

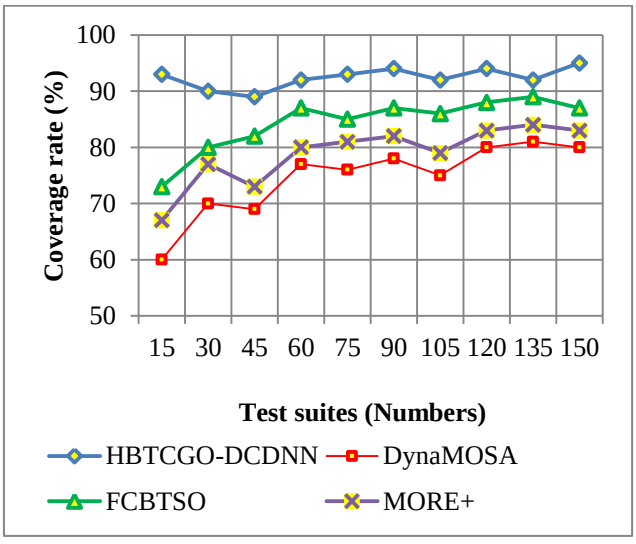

Fig. 9 Test suites Vs Coverage rate

Fig. 9 portrays the coverage rate concerning the number of test suites. The number of test suites is represented in the x direction, where the $y$value refers to the coverage rate. As shown in Fig. 9, the coverage rate of the HBTCGO-DCDNN is represented by the blue color lines. The coverage rate of the three existing methods [1][2] [3] is indicated by red, green, and violet respectively. As shown in the figure, the coverage rate is considerably increased in the proposed HBTCGO-DCDNN than the existing approaches. By applying the truncative cyclic uniformed gene optimization measures the fault coverage rate of each generated test suites. The test suites with a maximum fault coverage rate are taken as final test suites and other test suites are removed. In this way, the optimized selected number of test suites helps for increasing the coverage rate.

6. Conclusion

In this paper, the tests suites are created with reduce the time and cost, a novel technique HBTCGO-DCDNN is presented for test suite generation and optimization. The conventional method of software testing is taken more execution time and large number of test suites. In this paper, we present proposed HBTCGO-DCDNN technique. It is a multi-objective buffalo optimization implemented in the deep learning approach. We highlighted both the merits and

draw backs in this paper. If applied properly, the novel technique merits can help immensely. The main contribution of the work is to select the optimal test suites with lesser time and cost. An improved buffalo optimization technique is used to precisely discover the optimal test case and generate the test suites. Therefore, the optimal test suites are achieved by considering minimum time consumption. In order to eradicate the redundant test cases, the test suit reduction process are performed. Followed by, the optimal one that covers the more faults from the given software programs are discovered by using truncative cyclic uniformed gene optimization technique. This is helps to achieve higher test suite reduction rate and fault coverage rate. The comprehensive experimental evaluation is conducted with a webchess dataset. The quantitative performance analysis is conducted with HBTCGO-DCDNN and other existing methods. The systematic quantitative results indicate that our HBTCGO-DCDNN is better in terms of higher reduction rate by 19%, coverage rate by 18%, and lesser time consumption by 48% when compared to other related methods. This result demonstrates the proposed technique is effective and satisfactory.

In future, this work will be extended to explore various optimization methods to generate test data. This approach will ensure software quality with improved test suite reduction rate in an accurate manner.

References

[1] Annibale Panichella, Fitsum Meshesha Kifetew, Paolo Tonella, "Automated Test Case Generation as a Many-Objective Optimisation Problem with Dynamic Selection of the Targets", *IEEE Transactions on Software Engineering*, Volume 44, Issue 2, 2018, Pages 122 – 158.

[2] Arun Prakash Agrawal, Ankur Choudhary, Arvinder Kaur & Hari Mohan Pandey, "Fault coverage-based test suite optimization method for regression testing: learning from the mistakes-based approach", *Neural Computing and Applications*, Springer, Volume 32, 2020, Pages 7769–7784.

[3] Alessandro Marchetto, Giuseppe Scanniello, Angelo Susi, "Combining Code and Requirements Coverage with Execution Cost for Test Suite Reduction", *IEEE Transactions on Software Engineering*, Volume 45, Issue 4, 2019, Pages 363 – 390.

[4] Abdullah B. Nasser,Kamal Z. Zamli,AbdulRahman A. Alsewari,Bestoun S. Ahmed, "Hybrid flower pollination algorithm strategies for t-way test suite generation*", PLoS ONE*, Volume 13, Issue 5, Pages 1-24.

[5] Neha Gupta, Arun Sharma, Manoj Kumar Pachariya, "Multi-objective test suite optimization for detection and localization of software faults", *Journal of King Saud University - Computer and Information Sciences*, Elsevier, 2020, Pages 1-13.

[6] Rosziati Ibrahim, Ammar Aminuddin Bani Amin, Sapiee Jamel, Jahari Abdul Wahab, EPiT: A Software Testing Tool for Generation of Test Cases Automatically", *International Journal of Engineering Trends and Technology* (IJETT), Volume 68, Issue 7, 2020, Pages 8-12.

[7] Divya Taneja, Rajvir Singh, Ajmer Singh, Himanshu Malik, "A Novel technique for test case minimization in object-oriented testing", *Procedia Computer Science, Elsevier*, Volume 167 2020, Pages 2221–2228.

[8] Dharmveer Kumar Yadav and Sandip Dutta, "Regression test case selection and prioritization for object-oriented software", *Microsystem Technologies*, Springer, Volume 26, 2020, Pages 1463-1477.

[9] AutO. ÖrsanÖzener and Hasan Sözer, "An effective formulation of the multi-criteria test suite minimization problem", *Journal of Systems and Software*, Elsevier, Volume 168, 2020, Pages 1-12.

[10] Handing Wang, Yaochu Jin, John Doherty, "A Generic Test Suite for Evolutionary Multifidelity Optimization", *IEEE Transactions on Evolutionary Computation*, Volume 22, Issue 6, 2018, Pages 836 – 850.

[11] Wei Zheng, Xiaoxue Wu, Shichao Cao, Jun Lin, "MS-guided many-objective evolutionary optimization for test suite minimization", *IET Software*, Volume 12, Issue 6, 2018, Pages 547 – 554.

[12] Akram Kalaee and Vahid Rafe, "Model-based test suite generation for graph transformation system using model simulation and search-based techniques", *Information and Software Technology*, Elsevier, Volume 108, 2019, Pages 1-2.

[13] Yoo-Min Choi and Dong-Jin Lim, "Model-Based Test Suite Generation Using Mutation Analysis for Fault Localization", *Applied Science*, Volume 9, Issue 17, 2019, Pages 1-24.

[14] Abha Maru, Arpita Dutta, K. Vinod Kumar & Durga Prasad Mohapatra, "Software fault localization using BP neural network based on function and branch coverage", *Evolutionary Intelligence*, Springer, 2019, Pages 1-18.

[15] Andrea Arcuri, "Test suite generation with the Many Independent Objective (MIO) algorithm", *Information and Software Technology*, Elsevier, Volume 104, 2018, Pages 195-206.

[16] Xiaoan Bao, Zijian Xiong, Na Zhang, Junyan Qian, Biao Wu, Wei Zhang, "Path-oriented test cases generation based adaptive genetic algorithm", *PLoS ONE*, Volume 12, Issue 11, 2017, Pages 1-17.

[17] Haifeng Wang, Bin Du, Jie He, Yong Liu, Xiang Chen, "IETCR: An Information Entropy-Based Test Case Reduction Strategy for Mutation-Based Fault Localization", *IEEE Access*, Volume 8, 2020, Pages 124297 – 124310.

[18] Pedro Delgado-Pérez and Inmaculada Medina-Bulo, "Search-based mutant selection for efficient test suite improvement: Evaluation and results", *Information and Software Technology*, Elsevier, Volume 104, 2018, Pages 130-143.

[19] Warda Elkholy, Mohamed El-Menshawy, Jamal Bentahar, Mounia Elqortobi, Amine Laarej, Rachida Dssouli, "Model checking intelligent avionics systems for test cases generation using multi-agent systems", *Expert Systems With Applications*, Elsevier, Volume 156, 2020, Pages 1-19.

[20] Sonali Pradhan, Mitrabinda Ray, Santosh Kumar Swain, "Transition coverage based test case generation from state chart diagram", *Journal of King Saud University - Computer and Information Sciences*, Elsevier, 2019, Pages 1-1.

[21] Anbunathan R, Anirban Basu, "Basis Path Based Test Suite Minimization Using Genetic Algorithm", International Journal of Intelligent Systems and Applications (IJISA), Vol.10, No.11, pp.36-49, 2018. DOI: 10.5815/ijisa.2018.11.05.

[22] Abhinandan H. Patil, Neena Goveas, Krishnan Rangarajan,"Regression Test Suite Prioritization using Residual Test Coverage Algorithm and Statistical Techniques", International Journal of Education and Management Engineering(IJEME), Vol.6, No.5, pp.32-39, 2016.DOI: 10.5815/ijeme.2016.05.04.

[23] Abhinandan H. Patil, Neena Goveas, Krishnan Rangarajan,"Regression Test Suite Execution Time Analysis using Statistical Techniques", International Journal of Education and Management Engineering (IJEME), Vol.6, No.3, pp.33-41, 2016.DOI: 10.5815/ijeme.2016.03.04.

Authors' Profiles

T.Ramasundaram is currently a Ph.D. candidate in the Department of Computer Science, Periyar University, Salem, Tamilnadu, India. He received his B.Sc., M.Sc., and M.Phil. degrees in Computer Science from the same university, in 2003, 2005, and 2009 respectively. His current research interest is Software Engineering.

Dr V.SANGEETHA received her Ph.D. degree in computer science from the Mother Teresa Women's University, Kodaikanal, Tamilnadu, India, in 2014. She received her M.Sc. and M.Phil. degrees in computer science from the Periyar University, Salem, Tamilnadu, India, in 2001 and 2004 respectively. She is currently an Assistant Professor and Head, Govt Arts and Science College, Pappireddipatti, Dharmapuri, Tamilnadu, India. Her research interests include Software Engineering, Data Mining, and Compiler Design.

How to cite this paper: T. Ramasundaram, V. Sangeetha, "Hybridization of Buffalo and Truncative Cyclic Gene Deep Neural Network-based Test Suite Optimization for Software Testing ", International Journal of Modern Education and Computer Science(IJMECS), Vol.14, No.4, pp. 43-56, 2022.DOI: 10.5815/ijmecs.2022.04.04