ISSN 2515-8260 Volume 07, Issue 09, 2020

ADAPTIVE DIMENSIONAL PARTICLE SWARM OPTIMIZATION BASED HYPER BASIS FUNCTION NEURAL NETWORK CLASSIFICATION FOR SOFTWARE FAILURE CAUSE PREDICTION

Mr.P.SARAVANAN[1], Dr.V.SANGEETHA.,M.Sc.,Mphil.,Ph.D[2]

[1]*Asst.Professor, Department of computer science, Govt. Arts College, Dharmapuri, Tamilnadu,*

[2]*Asst.Professor, Department of computer science, Govt. Arts College, Pappireddipatti, Tamilnadu*

ABSTRACT

In software engineering, Detection of software root cause failure is a considerable issue to be resolved for increasing the reliability. Few research works are introduced for predicting the software failure causes with help of diverse classification algorithm. However, False Positive Rate (FPR) of failure detection process was higher. Therefore, a novel software failure cause prediction model called Adaptive Dimensional Particle Swarm Optimization Based Hyper Basis Function Neural Network (ADPSO-HBFNN) Model is proposed to increase the software reliability through predicting the root cause of software failure at an early stage. ADPSO-HBFNN Model initially gets number of event log files as input. Next, ADPSO-HBFNN Model applies Hyper Basis Function Neural Networks (HBFNNs) for discovering the software fault root cause by means of classifying the event log files. Subsequently, ADPSO-HBFNN Model applies Adaptive Dimensional Search Based Particle Swarm Optimization (ADS-PSO) algorithm where it considers the cost sensitive factor such as expected cost of software failure misclassification. The ADS-PSO algorithm lessen mean square error (MSE) during the learning process by optimizing the weights of network. From that, ADPSO-HBFNN Model correctly find outs the root cause of software failure with higher accuracy. Simulation outcome of ADPSO-HBFNN Model increase the accuracy and lessen time required for software fault root cause prediction as compared to conventional works.

Keywords: Adaptive Dimensional Search, Event Log Files, Hyper Basis Function Neural Networks, Particle Swarm Optimization, Root Cause, Software Failure

1. INTRODUCTION

Software root cause analysis finds the faults that originate system application crashes. In modern software system, logs are utilized in order to record system events at runtime. Software defect prediction process finds out possible fault and thereby enhances the quality. There are many research works have been designed for software root cause failure analysis with help of different techniques. However, existing techniques does not provided higher accuracy for both root cause and software defect discovery. Therefore, a novel ADPSO-HBFNN Model is introduced by using the HBFNNs and ADS-PSO algorithm.

To perform event-based failure prediction with minimal time, Artificial neural network (ANN) model was employed in [1]. However, software cause prediction accuracy was very lower. In [2], Hierarchical online failure prediction approach called Hora was designed to discover failures. But, expected cost of software failure misclassification was more.

Ant Colony Optimization (ACO)-based classification technique was introduced in [3] to predict erroneous software modules. However, TC of software cause detection was more. For software fault prediction, a hybrid one-class rule learning approach was introduced in [4]. But, FPR of software fault discovery was not considered.

In [5], Bayesian Regularization (BR) technique was designed to discovering the software faults and lessens cost of software testing. However, accuracy of root cause and software defect detection was poor. A survey of diverse classification techniques designed for accomplishing software fault detection was analyzed in [6].

To enhance the detection performance of software faults with minimal time, Quad Tree-based K-Means algorithm was presented in [7]. But, fault prediction accuracy was very lower. A review of varied machine learning algorithms intended for carried outing the software fault discovery was presented in [8].

Finding the effort interrelated with discovering software errors were presented in [9]. But, TC during the fault detection process was remained open issue. In order to addresses the significant issues in software failure detection, a Mamdani type fuzzy inference system (FIS) was designed in [10].

To resolve the aforementioned conventional issues, ADPSO-HBFNN Model is presented. Contribution of ADPSO-HBFNN Model is described below,

- ✓ The HBFNNs employed in ADPSO-HBFNN Model for better classification performance when taking a large number of event log files as input. Hence, HBFNNs used in ADPSO-HBFNN.

- ✓ ADS-PSO algorithm varies from nature-inspired metaheuristic techniques because it does not employ any metaphor as underlying principle for implementation. Besides to that, ADS-PSO algorithm provides accurate software failure cause detection. Also, ADS-PSO algorithm updates the search dimensionality ratio for rapid and reliable convergence to optimum.

- ✓ ADS-PSO algorithm includes benefits such as quick convergence, a small number of setting parameters, and easy implementation. For this reason, ADS-PSO algorithm used in ADPSO-HBFNN Model to solve expected cost of misclassification problem in software failure cause identification process with enhanced accuracy.

The residual structure of article is follows: ADPSO-HBFNN Model is described in Section 2. Experimental settings and results are discussed in Section 3 and Section 4. Literature survey is demonstrated in Section 5. The paper is concluded in Section 6.

2. ADAPTIVE DIMENSIONAL PARTICLE SWARM OPTIMIZATION BASED HYPER BASIS FUNCTION NEURAL NETWORK MODEL

An ADPSO-HBFNN Model is designed in order to effectively find the root cause of software failures by using the event logs. The ADPSO-HBFNN Model is proposed with help of HBFNNs and ADS-PSO algorithm. ADPSO-HBFNN Model gives good performance in different application domains. HBFNN is a variant of three-layer feed forward neural networks. The advantage of employing HBFNNs in ADPSO-HBFNN Model is due to its faster convergence. To lessen the time taken for convergence, the weights of HBFNNs are optimized with assist of ADS-PSO algorithm. The HBFNNs model performs software failure cause prediction process to get better accuracy with faster convergence.

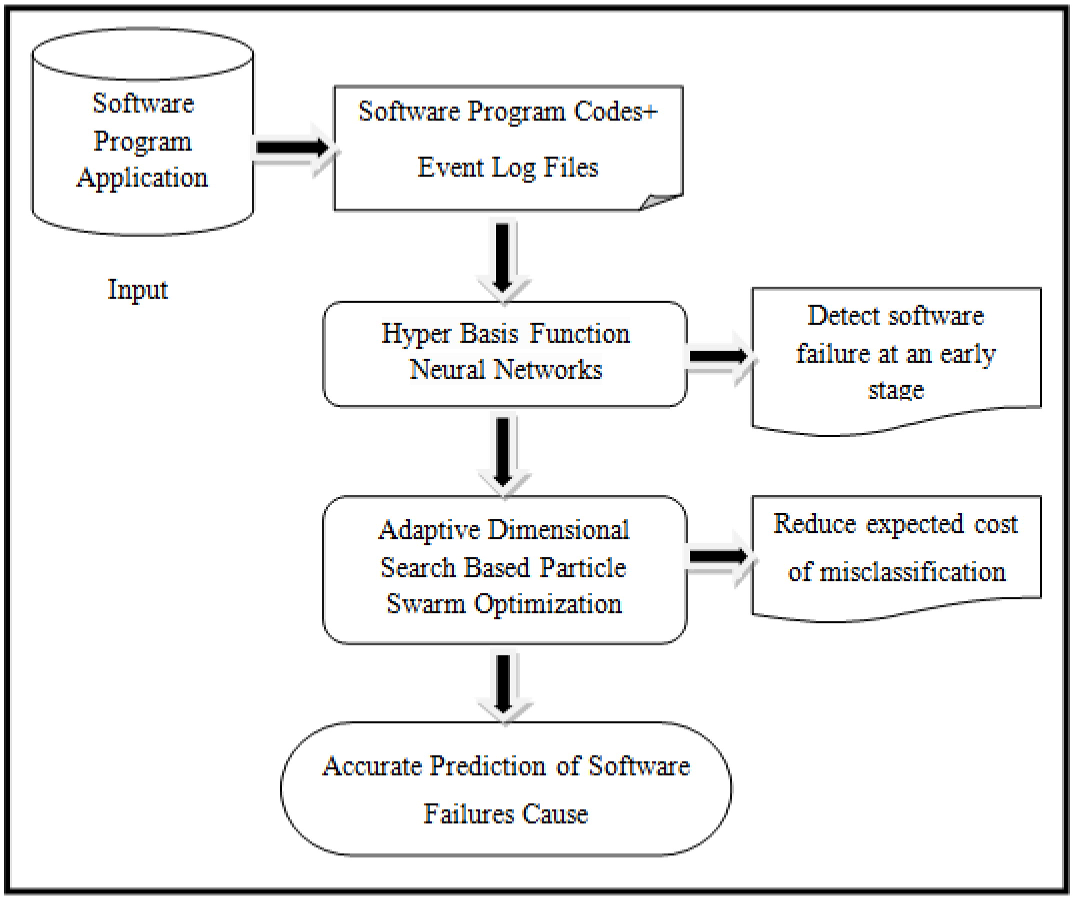

Software Program Application
Input
Software Program Codes+ Event Log Files
Hyper Basis Function Neural Networks
Detect software failure at an early stage
Adaptive Dimensional Search Based Particle Swarm Optimization
Reduce expected cost of misclassification
Accurate Prediction of Software Failures Cause

Figure 1 Architecture Diagram of ADPSO-HBFNN Model for Software Failure Cause Prediction

The overall process of ADPSO-HBFNN Model is illustrated in Figure 1 to attain better software failure cause detection accuracy with time. At first, ADPSO-HBFNN acquires software program codes and event log files from Blue Gene/P Intrepid system as input. After obtaining the event log files, ADPSO-HBFNN Model employs HBFNNs to discover cause of software failures. Followed by, ADPSO-HBFNN Model used ADS-PSO algorithm lessen the expected cost of misclassification during the software failure cause prediction through optimizing the weights of the HBFNNs. As a result, ADPSO-HBFNN Model exactly finds software failure causes.

2.1 Hyper Basis Function Neural Networks

In ADPSO-HBFNN Model, HBFNNs algorithm is a kind of feed-forward neural networks. The HBFNNs obtains a number of event log files as input. The HBFNNs contains input, hidden and output layer in order to classify each event log files with higher accuracy. The structure diagram of HBFNNs is presented in below Figure 2.

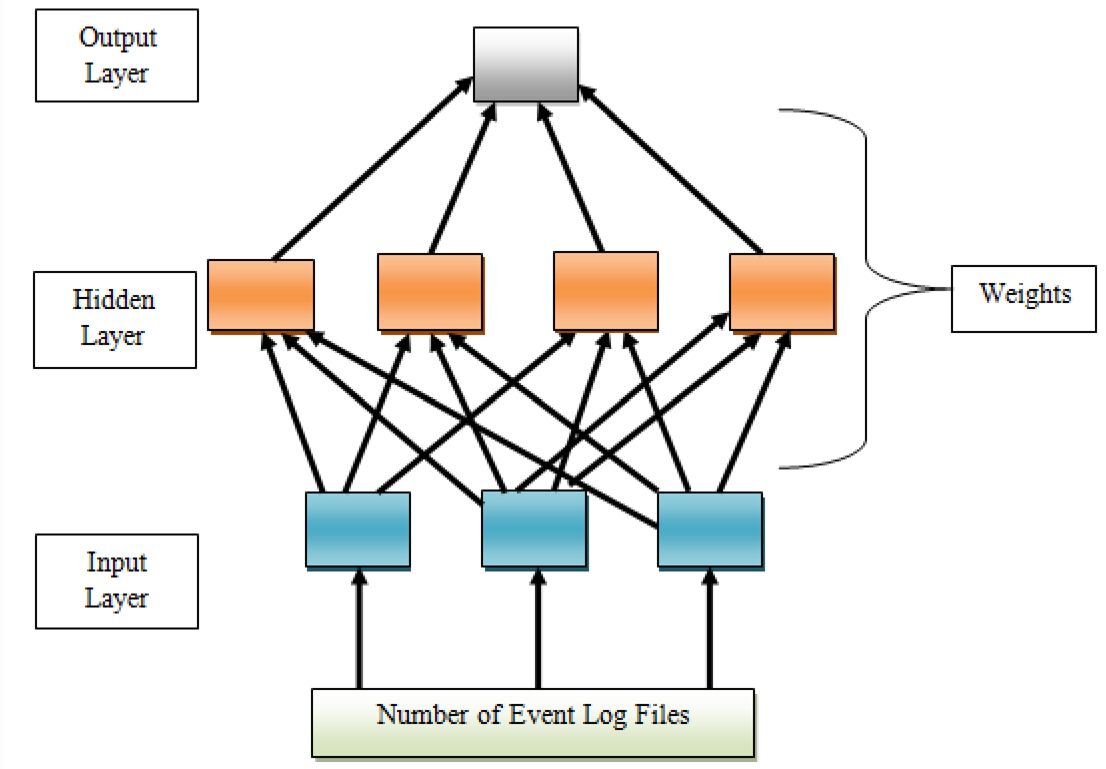


Figure 2 Structural Diagrams of HBFNNs for Software Failure Cause Detection

As illustrated in the above structural diagram 2, HBFNNs includes of three layers. Input layer obtains number of event log files and sends to the hidden layer. Hidden layer scrutinizes input event log files using activation function and generates the classification result to the third layer called output layer. Every interconnection in HBFNNs contains a strength called weight. The weight is referred by a number. The HBFNNs learns input event log files by adjusting the weights of each neuron to get higher classification accuracy with minimal time.

Let us consider number of event log files in given dataset are denoted as '$l_i = l_1, l_2, .., l_n$'. In HBFNNs, Gaussian activation function finds the relationship between the input event log files. The output of activation function is either '0' or '1'. From that, 'F' output in hidden layer is mathematically estimated as follows,

$$\mathrm{F}(l_i) = \frac{1}{\sqrt{2\pi\sigma}} e^{\frac{-(l_i-m)^2}{2v^2}} \quad (1)$$

From the above mathematical equation (1), 'l_i' signifies input event log files. Here, 'm' and 'v' represent the mean and variance value between the event log files and software failure conditions. The output of Gaussian activation function '1' denotes that there is a root cause of software failure is found whereas '0' indicates that there is a root cause of software failure is not found. For each the obtained result, then HBFNNs computes error rate '$\beta(t)$' using below,

$$MSE = \beta(t) = \frac{1}{n}\sum_{i=1}^{n}(O_R - O_R)^2 \qquad (2)$$

From (2), HBFNNs estimates error rate of each classification result of input event log files whereas 'MSE' represents mean squared error. Here, 'O_R' is a target result, 'O_R' is an actual result. Subsequently, the HBFNNs update the weights on network based on calculated mean squared error. The processes of HBFNNs are continual until the mean squared error value is very minimal to accurately classify input event log files.

The goal of HBFNNs is to lessen MSE by optimizing the weights of network. The MSE determined is back propagated and weights are tuned to lessen the error with assist of ADS-PSO algorithm.

2.2 Adaptive Dimensional Search Based Particle Swarm Optimization

In ADPSO-HBFNN Model, error adjustments and tuning for optimal weights are performed by considering a new objective function i.e. the cost-sensitivity for effective software failure cause prediction. ADS-PSO algorithm at first initialize swarm population by considering the different number of random weights on HBFNNs. In ADS-PSO algorithm, initial populations of particles (i.e. number of random weights) are initialized using below mathematical formulation,

$$a_i = a_1, a_2, \ldots a_n \qquad (3)$$

From the above mathematical equation (3), 'a' refers a number of random weights on HBFNNs. Best solution of previous population create number of candidate solutions in current population. The ADS-PSO is based on updating search dimensional rate (SDR) metric. SDR is calculated as percentage of design variables which are perturbed when produce a candidate solution from current best population. From that, search dimensional rate is mathematically determined using below,

$$SDR = \frac{X_p}{X_d} \qquad (4)$$

From (4), 'X_p' is the number of design variables perturbed to create new solution and 'X_d' denotes total number of design variables.

Further, cost parameters in ADS-PSO algorithm are expected cost of misclassification and its normalized value. These cost-sensitive factors are assumed based on false positive error cost and false negative error cost. The objective function of cost-sensitive HBFNNs to be minimized by ADS-PSO which is obtained as follows,

$$OF \leftarrow Min_{NECM} = F_{PR} + p_{NDP} + \frac{C_{FN}}{C_{FP}} \times F_{NR} \times p_{DP} \quad (5)$$

From equation (5), '$NECM$' indicates normalized expected cost of misclassification whereas 'F_{PR}' is the false positive rate, 'F_{NR}' is false negative rate, 'C_{FP}' indicates cost pertaining to false positive error. Here, 'C_{FN}' refers the cost pertaining to false negative error and 'p_{NDP}' and 'p_{DP}' are percentage of non-defect-prone modules and defect-prone modules. ADS-PSO algorithm updates position and velocity of particles to discover the optimal weights.

Consider 'i' is a number of particles i.e. '$i = 1,..,n$'. Here, '$\varphi_i(t)$' signifies the position of particle 'in search space at time t. The position of particle is varied based on velocity '$\delta_i\,(t)$' to current position. Position of particles is updated as below,

$$\varphi_i\,(t+1) = \varphi_i\,(t) + \delta_i(t+1) \quad (6)$$

From (6), '$\varphi_i(t+1)$' is updated position and '$\varphi_i\,(t)$' indicates current position of particles and adjusted velocity '$\delta_i(t+1)$'. Subsequently, velocity of particle is updated as below,

$$\delta_i\,(t) = \omega_t\delta_i\,(t-1) + b_1k_1(pbest\,(t) - \varphi_i\,(t-1) + b_2k_2\,(gbest\,(t) - \varphi_i\,(t-1)) \quad (7)$$

From (7), 'δ_i' designate the particle velocity, 'φ_i' is position of current particle , '$k_{1,}k_2$' represent random number between 0 and 1. Here, 'b_1, b_2' signifies the acceleration factors and 'ω_t' designates weight factor. The current position of successful particles is updated when position and velocity of particle is updated. For each iteration, ADS-PSO algorithm determines the optimal weights through decreasing the MSE in order to accurately predicting the software failure causes with minimal time consumption.

```
// Adaptive Dimensional Particle Swarm Optimization Based Hyper Basis Function
Neural Network Algorithm
Input: Number of event log files '$l_i = l_1, l_2, \ldots, l_n$' ,
Output: Improved Accuracy for Software Failure Cause Prediction
Step 1: Begin
Step 2:   Initialize network with random weights
Step 3:   While ('$\beta(t)$' is minimal) do
Step 4:       For each input event log files '$l_i$' at the input layer
Step 5:           Input layer forward received event log files '$l_i$' to hidden layer
Step 6:             Hidden layer apply activation function using (1)
Step 7:             Hidden layer forwards discovered result to the output layer
Step 8:           Output layer gives classification result
Step 9:       End for
Step 10:      Determine error rate '$\beta(t)$' using (2)
Step 11:      Optimize weights on network using ADS-PSO algorithm
Step 12:  End while
Step 13:  If (F($l_i$) = 1) then
Step 14:      Root Cause Of Software Failure Is Found
Step 15:  Else
Step 16:      Root Cause Of Software Failure Is Not Found
Step 17:  End If
```

Algorithm 1 Adaptive Dimensional Particle Swarm Optimization Based Hyper Basis Function Neural Network

The processes of ADPSO-HBFNN Model is presented in Algorithm 1 for precisely predicts the cause of given software failure with lower TC. Thus, ADPSO-HBFNN Model provides enhanced accuracy for discovering cause of given software failure application when compared to conventional works.

3. EXPERIMENTAL SETTINGS

ADPSO-HBFNN Model is implemented in java language with event log files taken from Blue Gene/P Intrepid system [21]. The input event log files are collected from period of 6 months on Blue Gene/P Intrepid system. Effectiveness of ADPSO-HBFNN Model is determined in accuracy, TC and FPR. Result of ADPSO-HBFNN Model is compared with [1] and [2].

4. RESULTS

The result analysis of ADPSO-HBFNN Model is described and compared with existing [1] and [2]. ADPSO-HBFNN Model is evaluated with following metrics with the assist of tables and graphs.

4.1 Experimental measure of Accuracy

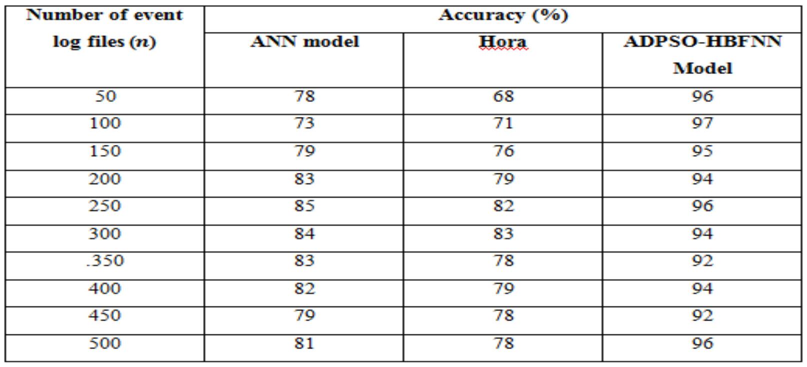
Accuracy 'A' is calculated as ratio of number of event log files which are correctly detected as cause or not to total number of event log files taken as input. Thus, accuracy is mathematically obtained as follows,

$$A = \frac{X_{AD}}{n} * 100 \qquad (8)$$

From (8), 'n' indicates a total number of event log files whereas 'X_{AD}' point outs number of event log files that are accurately detected. The accuracy is calculated in percentage (%).

When carried outing an experimental evaluation using 100 event log files from an input dataset, proposed ADPSO-HBFNN Model achieves 97 % accuracy whereas existing ANN model [1] and Hora [2] acquires 73 % and 71 % respectively. Accordingly, it is expressive that the accuracy of software failures causes detection using proposed ADPSO-HBFNN Model is higher than other traditional works ANN model [1] and Hora [2]. The experimental result of accuracy is obtained during the processes of software failure prediction are demonstrated in Table 1.

Table 1 Tabulation for Accuracy



Number of event log files (n)	Accuracy (%)		
	ANN model	Hora	ADPSO-HBFNN Model
50	78	68	96
100	73	71	97
150	79	76	95
200	83	79	94
250	85	82	96
300	84	83	94
.350	83	78	92
400	82	79	94
450	79	78	92
500	81	78	96

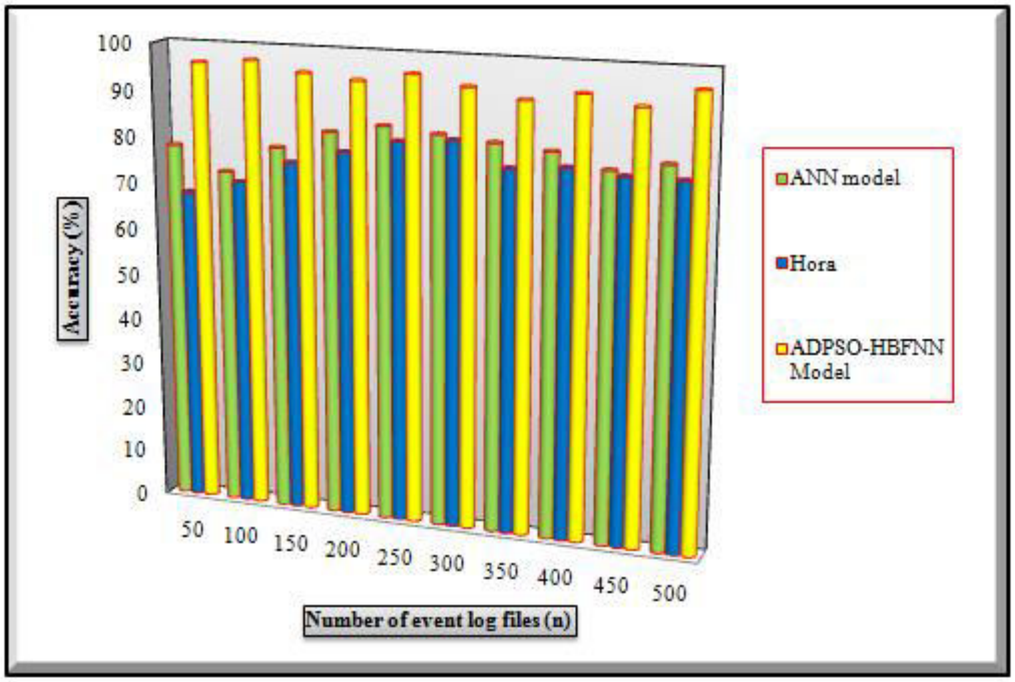


Figure 3 Graphical Result Analysis of Accuracy versus Different Number of Event Log Files

The result of accuracy is depicted in figure 3 with number of event log files ranges from 50 to 500 by using three techniques. From figure 3, ADPSO-HBFNN Model attain higher accuracy for predicting software failure causes as compared to existing [1] and Hora [2]. This is because of application of HBFNNs and ADS-PSO algorithm in proposed ADPSO-HBFNN Model. This assists to enhance the number of event log files exactly detected as failure cause or not. Therefore, proposed ADPSO-HBFNN Model increases the accuracy of failure causes discovery by 17 % and 23 % than the existing [1] and [2].

4.2 Experimental measure of Time Complexity

'TC' calculates the amount of time required to identify the software failure cause via classifying input event log files. The TC is mathematically expressed as follows,

$$TC = n * T\ (CSLF) \tag{9}$$

From the above mathematical equation (9), the TC of software failure cause prediction is measured. Here, 'n' signify the number of event log files and '$T\ (CSLF)$' denotes the time taken for classifying a single event log file. The TC of software application failure cause prediction is determined in milliseconds (ms).

When accomplishing an experimental process using 350 event log files from an given dataset, TC of ADPSO-HBFNN Model is 88 ms whereas TC of existing [1] and [2] is 109 ms and 105 ms. For that reason, the TC of ADPSO-HBFNN Model is very minimal than the existing

[1] and [2]. The performance result of TC is acquired during the processes of software failure detection is presented in Table 2.

Table 2 Tabulation for Time Complexity

Number of event log files (*n*)	Time Complexity (ms)		
	ANN model	Hora	ADPSO-HBFNN Model
50	21	23	19
100	35	37	32
150	51	54	47
200	64	68	58
250	75	85	70
300	84	99	78
350	109	105	88
400	124	120	92
450	135	131	99
500	150	145	105

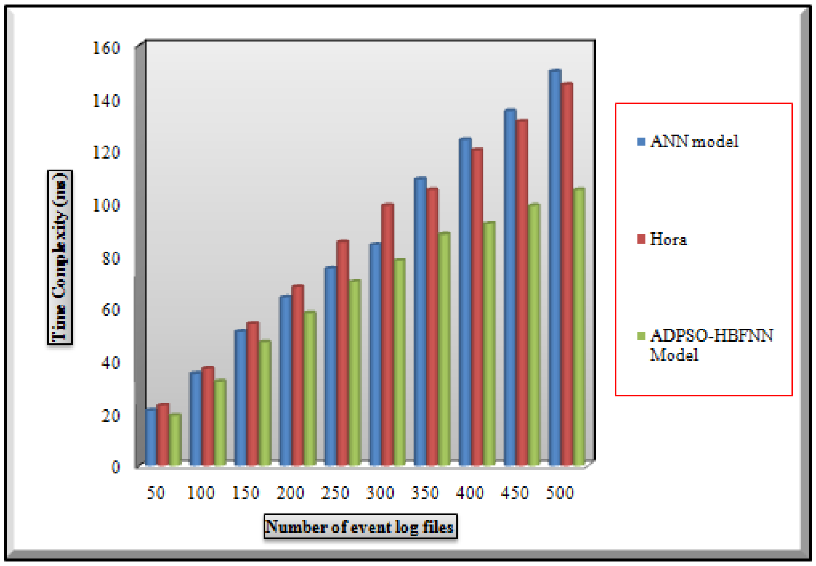


Figure 4 Graphical Result Analysis of Time Complexity versus Different Number of Event Log Files

Figure 4 shows experimental measure of TC using three techniques. From figure, ADPSO-HBFNN Model affords minimal TC with increasing number of event log files as input as compared to ANN model [1] and Hora [2]. This is due to application of HBFNNs and ADS-PSO algorithm in proposed ADPSO-HBFNN Model. This assists to lessen the time for discovering software failure cause by means of classifying input event log files. Therefore, proposed ADPSO-HBFNN Model minimizes the TC of software application by 15 % and 19 % as compared to conventional ANN model [1] and Hora [2].

4.3 Experimental measure of False Positive Rate

'FPR' is calculated as ratio of number of event log files which are incorrectly detected as cause or not to total number of event log files. FPR is calculated as follows,

$$FPR = \frac{X_{ID}}{n} * 100 \qquad (10)$$

From (10), 'n' is the total number of event log files whereas 'X_{ID}' point outs number of event log files that are inaccurately detected. The FPR of software application failure cause prediction is estimated in percentage (%).

When conducting an experimental work by taking 450 event log files from an input dataset, proposed ADPSO-HBFNN Model acquires 8 % FPRwhereas traditional ANN model [1] and Hora [2] attains 21 % and 23 % respectively. Therefore, the FPR of software failures causes' detection process using proposed ADPSO-HBFNN Model is very lower than other conventional works ANN model [1] and Hora [2]. Result of FPR determined during the processes of software failure discovery according to diverse number of event log files using three methods is demonstrated in Table 3.

Table 3 Tabulation for False Positive Rate

Number of event log files (n)	False Positive Rate (%)		
	ANN model	Hora	ADPSO-HBFNN Model
50	22	32	4
100	27	29	3
150	21	24	5
200	18	22	7
250	15	18	4
300	16	17	6
.350	17	22	8
400	19	22	6
450	21	23	8
500	19	22	4

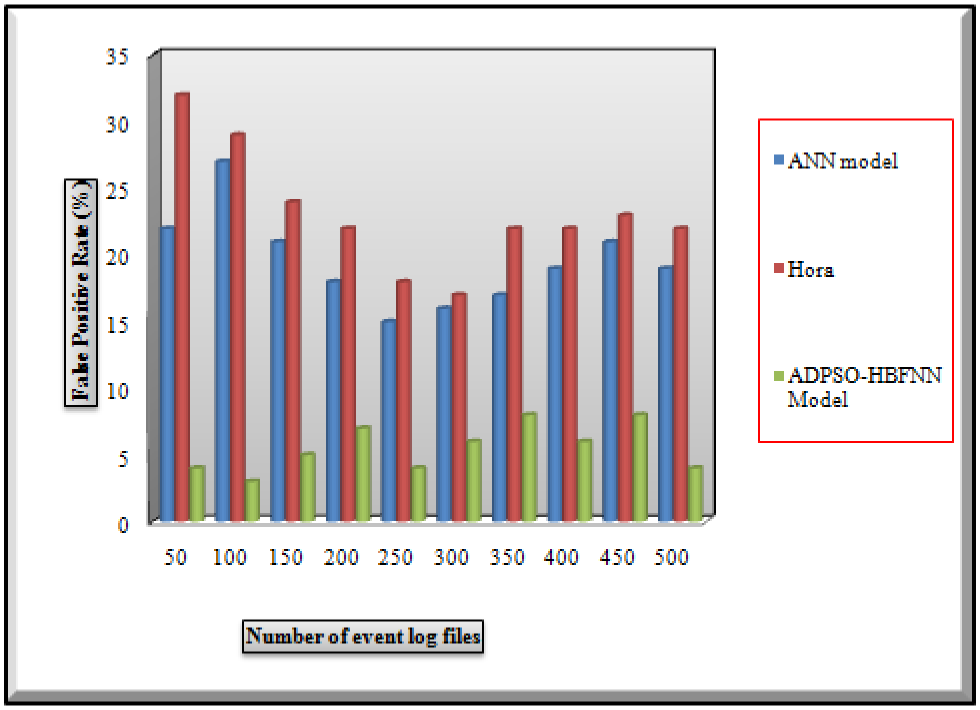


Figure 5 Graphical Result Analysis of False Positive Rate versus Different Number of Event Log Files

Result of FPR is portrayed in figure 5 with number of event log files ranges from 50 to 500. From figure 5, the ADPSO-HBFNN Model affords FPR minimal as compared to ANN model [1] and Hora [2]. This is due to application of HBFNNs and ADS-PSO algorithm in proposed ADO-HBFNN model. This assists to reduce the event log files that are incorrectly detected as cause or not. Therefore, proposed ADPSO-HBFNN Model minimizes the FPR by 71 % and 75 % as compared to [1] and [2].

5. LITERATURE SURVEY

Analytics-driven testing (ADT) was accomplished in [11] to discover types of software system failures with lower error rate. But, failure detection accuracy using ADT was not adequate. A cluster based fault prediction classifiers were employed in [12] that gets better fault detection performance with minimal time complexity. However, FPR during the fault detection process was more.

A Combined-Learning Based Framework was introduced in [13] to achieve enhanced accuracy and minimal amount of time for software fault prediction. But, misclassification problem was not solved. A fuzzy rule based algorithm was utilized in [14] for accurate discovery of faults in software during the software testing process. However, the amount of time consumed for predicting the software failure was more.

Diversity Based Oversampling method was employed in [15] with the aim of resolving the class imbalance problems in defect prediction. But, fault detection accuracy was not improved. An attention-based recurrent neural network was used in [16] to get better software reliability. However, TC was remained open issue.

An Ordinal Classification method was introduced in [17] with objective of minimizing the FPR of software bug prediction. Naive Bayes classifier was presented in [18] for detecting the software faults with lower computational complexity. However, fault detection accuracy was lower.

An automated approach was implemented in [19] with the goal of reducing the maintenance time and cost of software fault detection. But, accuracy using this approach was minimal. A Log-logistic testing effort function model was employed in [20] for carried outing the fault detection and correction. However, cost of software fault discovery was more.

6. CONCLUSION

The ADPSO-HBFNN Model is proposed with the goal of increasing software reliability by discovering the root cause of software failure with minimal misclassification error. ADPSO-HBFNN Model enhances the ratio of number of event log files correctly detected. Moreover, the proposed ADPSO-HBFNN Model minimizes the amount of time needed to discover the software failure cause through categorizing input event log files. Proposed ADPSO-HBFNN Model decreases number of event log files that are wrongly predicted. The experimental result shows that ADPSO-HBFNN Model provides better performance in terms of accuracy and TC as compared to conventional works.

REFERENCES

[1] Michael Borkowski, Walid Fdhila, Matteo Nardelli, Stefanie Rinderle-Ma, Stefan Schulte, “Event-based failure prediction in distributed business processes”, Information Systems, Elsevier, Volume 81, Pages 220-235, March 2019

[2] Teerat Pitakrat, Dusan Okanovic, Andrévan Hoorn, Lars Grunske, “Hora: Architecture-aware online failure prediction”, Journal of Systems and Software, Elsevier, Volume 137, Pages 669-685, March 2018

[3] Olivier Vandecruys, David Martens, Bart Baesens, Christophe Mues, Manu De Backer, Raf Haesen, “Mining software repositories for comprehensible software fault prediction models”, Journal of Systems and Software, Elsevier, Volume 81, Issue 5, Pages 823-839, May 2008

[4] Yousef Abdi, Saeed Parsa, Yousef Seyfari, “A hybrid one-class rule learning approach based on swarm intelligence for software fault prediction”, Innovations in Systems and Software Engineering, Springer, Volume 11, Issue 4, Pages 289–301, December 2015

[5] Rohit Mahajan, Sunil Kumar Gupta, Rajeev Kumar Bedi, “Design of Software Fault Prediction Model Using BR Technique” Procedia Computer Science, Elsevier, Volume 46, Pages 849-858, 2015

[6] Santosh S. Rathore, Sandeep Kumar, “An empirical study of some software fault prediction techniques for the number of faults prediction”, Soft Computing, Springer, Volume 21, Issue 24, Pages 7417–7434, 2016

[7] Partha S. Bishnu, Vandana Bhattacherjee, "Software Fault Prediction Using Quad Tree-Based K-Means Clustering Algorithm", IEEE Transactions on Knowledge and Data Engineering, Volume 24, Issue 6, Pages 1146 – 1150, June 2012

[8] Ruchika Malhotra, "A systematic review of machine learning techniques for software fault prediction", Applied Soft Computing, Elsevier, Volume 27, Pages 504-518, February 2015

[9] Maggie Hamill, Katerina Goseva-Popstojanova, "Analyzing and predicting effort associated with finding and fixing software faults", Information and Software Technology, Elsevier, Volume 87, Pages 1-18, July 2017

[10] Ezgi Erturk, Ebru Akcapinar Sezer, "Software fault prediction using Mamdani type fuzzy inference system", International Journal of Data Analysis Techniques and Strategies, Volume 8, Issue 1, Pages 14 – 28, 2016

[11] Feras A. Batarseh, Avelino J. Gonzalez, "Predicting failures in agile software development through data analytics", Software Quality Journal, Springer, Pages 1–18, 2015

[12] Pradeep Singh and Shrish Verma, "An Efficient Software Fault Prediction Model using Cluster based Classification", International Journal of Applied Information Systems, Volume 7, Issue 3, Pages 35-41, May 2014

[13] Chubato Wondaferaw Yohannese, Tianrui Li, "A Combined-Learning Based Framework for Improved Software Fault Prediction", International Journal of Computational Intelligence Systems, Volume 10, Issue 1, Pages 647 – 662, 2017

[14] Subhashis Chatterjee, Bappa Maji, "A New Fuzzy Rule Based Algorithm for Estimating Software Faults in Early Phase of Development", Soft Computing, Springer, Volume 20, Issue 10, Pages 4023–4035, June 2015

[15] Kwabena Ebo Bennin, Jacky Keung, Passakorn Phannachitta, Akito Monden, Solomon Mensah, "MAHAKIL: Diversity Based Oversampling Approach to Alleviate the Class Imbalance Issue in Software Defect Prediction", IEEE Transactions on Software Engineering, Volume 44, Issue 6, Pages 534 – 550, June 2018

[16] Guisheng Fan, Xuyang Diao, Huiqun Yu, Kang Yang, and Liqiong Chen, "Software Defect Prediction via Attention-Based Recurrent Neural Network", Scientific Programming, Volume 2019, Article ID 6230953, Pages 1-14, 2019

[17] Elife Öztürk Kıyak, Kökten Ulaş Birant, Derya Birant, "An Ordinal Classification Approach for Software Bug Prediction", Dokuz Eylul University Faculty of Engineering Journal of Science and Engineering, Volume 21, Issue 62, Pages 533-544, 2019

[18] Bhekisipho Twala, "Predicting Software Faults in Large Space Systems using Machine Learning Techniques", Defence Science Journal, Volume 61, Issue 4, Pages 306-316, 2011

[19] Amjad A. Hudaib, Hussam N. Fakhouri, "An Automated Approach for Software Fault Detection and Recovery", Communications and Network, Volume 8, Pages 158-169, 2016

[20] Zafar Imam, Ishrat Jahan Ara and N. Ahmad, "Analysis of Software Fault Detection and Correction Processes with Log-Logistic Testing-Effort", Recent Advances in Mathematics, Statistics and Computer Science, Pages 549-560, 2016