Indonesian Journal of Electrical Engineering and Computer Science
Vol. 24, No. 3, December 2021, pp. 1804~1813
ISSN: 2502-4752, DOI: 10.11591/ijeecs.v24.i3.pp1804-1813

Multicriteria Cuckoo search optimized latent Dirichlet allocation based Ruzchika indexive regression for software quality management

R. Chennappan[1], Vidyaa Thulasiraman[2]
[1]Department of Computer Science, Periyar University, Salem, India
[2]Department of Computer Science, Government Arts and Science College for Women, Barugur, India

Article Info

Article history:

Received Dec 22, 2020
Revised Oct 15, 2021
Accepted Oct 27, 2021

Keywords:

Generative latent Dirichlet allocation model
Ruzchika indexive regression
Multicriteria reinforced Cuckoo search optimization
Software quality management

ABSTRACT

The paper presents the software quality management is a highly significant one to ensure the quality and to review the reliability of software products. To improve the software quality by predicting software failures and enhancing the scalability, in this paper, we present a novel reinforced Cuckoo search optimized latent Dirichlet allocation based Ruzchika indexive regression (RCSOLDA-RIR) technique. At first, Multicriteria reinforced Cuckoo search optimization is used to perform the test case selection and find the most optimal solution while considering the multiple criteria and selecting the optimal test cases for testing the software quality. Next, the generative latent Dirichlet allocation model is applied to predict the software failure density with selected optimal test cases with minimum time. Finally, the Ruzchika indexive regression is applied for measuring the similarity between the preceding versions and the new version of software products. Based on the similarity estimation, the software failure density of the new version is also predicted. With this, software error prediction is made in a significant manner, therefore, improving the reliability of software code and service provisioning time between software versions in software systems is also minimized. An experimental assessment of the RCSOLDA-RIR technique achieves better reliability and scalability than the existing methods.

Corresponding Author:

R. Chennappan
Department of Computer Science, Periyar University
Salem, Tamilnadu, India
Email: Chennappanphd@gmail.com

1. INTRODUCTION

Software quality prediction facilitates minimizing the software costs and it identifies the risk-prone software components. Software quality estimation is a process of focusing on software development efforts. Reliability analysis is a significant indicator for predicting software quality and it has important research significance in recent days to increase reliable software by quantitatively estimating the software reliability. Therefore, software reliability prediction models are extremely useful for testing engineers to take critical decisions, such as the testing resource allocation and software release time evaluation. To construct the software system more efficiently, novel recent techniques and methodologies are needed for predicting and estimating software reliability. To improve the reliability of software development, failure prediction using machine learning is the most significant area of interest.

The existing methods are described in the major issues such as failure to perform optimal test case selection, scalability performance was not considered, lesser software reliability, higher time consumption,

minimal scalability, the efficiency of the failure prediction was not improved, and failure to provide the accurate predictions. To overcome the issues, motivated by this fact, the novel reinforced Cuckoo search optimized latent Dirichlet allocation based Ruzchika indexive regression (RCSOLDA-RIR) technique is developed. The strengths of the proposed technique enhance the software quality management, reliability, and scalability, and reduces the service provisioning time. The main contribution and novelty of the proposed methodology are summarized as follows:

- The main contribution of the proposed RCSOLDA-RIR technique is introduced to enhance the software quality management for performing the component-based failure prediction. The contribution is achieved by Multicriteria reinforced Cuckoo search optimization technique, generative latent Dirichlet allocation (GLDA) model, and Ruzchika indexive regression.
- Multicriteria reinforced Cuckoo search optimization is applied to find the optimal test case with lesser cost and minimum time. The novelty involved in multicriteria reinforced Cuckoo search optimization is to apply the Multicriteria optimization function. According to the multicriteria function, the fitness is evaluated for all test cases. Depended on the fitness value, the test cases ranked and determine the current best solution to achieve higher scalability.
- Generative latent Dirichlet allocation model to analyze the component failure density of the given software product with the selected optimal test cases. It is used to determine the software failure probability of every component and topic with minimum service provisioning time.
- The novelty involved in Ruzchika Indexive Regression in RCSOLDA-RIR to find the similarities among two neighboring versions to predict the software quality. Ruzicka's similarity function introduces the similarity value between zero and one. When the similarity value is higher than the threshold, that is said to be the higher similarity. Otherwise, it is said to be less similar. Therefore, improving the reliability of software.

The rest of the paper is organized as follows: in section two, discusses the related works, and section three provides the methodology of research in detail, and section four describes the experimental evaluation of different techniques using the schoolmate dataset, the results and discussion are presented in section five, and the conclusion is provided in Section six.

2. RELATED WORKS

A hybrid Wolf Pack algorithm and particle swarm optimization (WPA-PSO) was developed in [1] to increase the software reliability. The designed algorithm failed to improve the efficiency of the failure prediction. A fuzzy PSO algorithm to train the artificial neural network (ANN) based software reliability model (FPSONNM) [2] was developed to increase the software reliability. But the optimal test case selection was not performed A multi-objective technique was introduced in [3] to select the minimal test suite. The designed technique failed to consider reliability estimation. Ant colony optimization (ACO) [4], antlion optimization (ALO) algorithm [5] were discussed. Differential search algorithm (DSA) was developed in [6] for optimal choice and allocation of shunt Flexible AC transmission systems (FACTS) device. An error iterative analysis model was developed in [7] to predict software failures. An artificial bee colony algorithm and particle swarm optimization (ABC-PSO) were designed in [8]. However, it failed to consider better scalability performance. A component-based software failure prediction was performed in [9] for reducing the complexity of the system. A generalized software reliability growth method was developed in [10] for detecting the software failure data.

A multi-criteria test case selection method [11] was introduced. However, it failed to choose the optimal test cases for detecting more faults. An optimized neuro-PSO-based software maintainability prediction model was introduced in [12] to optimize the software maintenance prediction result. Different machine learning algorithms were developed in [13] to increase the accuracy of software failure prediction. But the algorithms failed to perform the scalability analysis. The soft computing techniques were introduced in [14] for predicting software reliability. A new framework was developed in [15] for detecting the error in the software system. But the designed framework failed to accurately detect the error with the help of test cases. A novel canonical correlation analysis was performed in [16] for identifying the heterogeneous fault prediction. However, the correlation analysis failed to validate the generalization ability of software defect prediction. A median absolute deviation threshold-based spectral classifier was developed in [17] to increase the software quality. A new nonnegative sparse graph-based label propagation model was introduced in [18] to identify the software defects. However, efficient defect prediction performance was not achieved. A learning deep feature representation (LDFR) was developed in [19] to identify the software faults. The deep learning concept was introduced in [20]-[21] to improve the software reliability factors.

Autoregressive integrated moving average (ARIMA) and long short-term memory model (LSTM) were used in [22] to reduce the error rate. Logistic regression and random forest were introduced in [23] to

improve the accuracy. An enhanced software quality monitoring framework was developed in [24]. An ensemble oversampling method [25] and finding faults using ensemble learners (ELFF) [26] were presented to predict the defects. However, it failed to investigate the factors that may impact the defects in a new software version. Different software piracy techniques were introduced in [27] to resolve privacy in software development.

3. RESEARCH METHOD

The test case is a set of conditions or rules to test the quality and satisfying the user requirements. The combination of test cases collectively named test suites. When considering a large number of test cases, the system consumes more time for testing the software products. Accordingly, the quality of the software gets reduced. In this paper, a novel technique called RCSOLDA-RIR is introduced for improving the software quality by using test cases on open-source applications.

Figure 1 given illustrates the flow process of the proposed RCSOLDA-RIR to achieve better scalability and reliability. Initially, the different software program source codes are taken from the schoolmate dataset for testing the quality. The RCSOLDA-RIR technique performs component-based software quality development. The component is an independent piece of code that deals with the amount of functionality. Initially, the Multicriteria reinforced Cuckoo search optimization technique is applied for choosing the optimal test case for testing the given software program codes. The generative latent Dirichlet allocation (GLDA) model is applied in the RCSOLDA-RIR technique to measure components and the topic failure density with the selected optimal test cases. Finally, Ruzchika Indexive Regression is applied to predict component failures. This process of the RCSOLDA-RIR technique is explained in the following subsections.

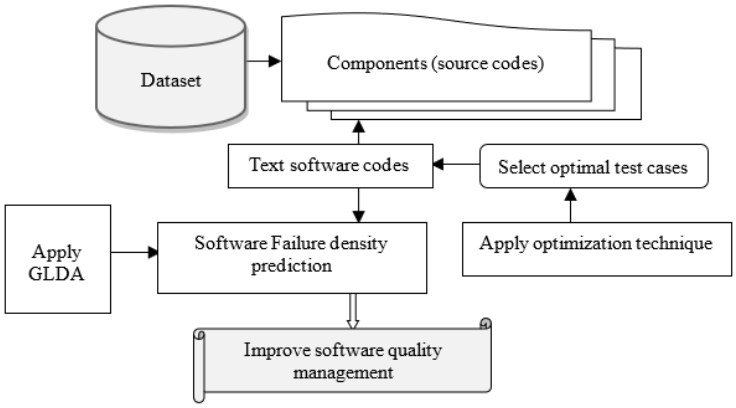

Figure 1. Flow process of proposed RCSOLDA-RIR technique

3.1. Multicriteria reinforced Cuckoo search optimization

Initially, the source codes are collected from the different version management systems. In software engineering, a component is an independent part of source codes. After collecting the components, the defects of each source code line are identified by testing the quality of the software. The testing of source code lines in a given software program application is performed through the optimal test suites. A test suite comprises the number of test cases. Among the multiple test cases, an optimal test case is selected to test the quality of the software program. The proposed RCSOLDA-RIR technique uses the multicriteria reinforced Cuckoo search optimization to choose the optimal test case. In existing, Cuckoo search optimization is used to solve the optimization issues and discover the best solution. Multicriteria denote an optimization technique that considers the multiple objective functions. Multiple objective functions are used to minimize the cost and time. Reinforced means strengthen the optimization algorithm by using the multicriteria function. Reinforced Cuckoo search optimization is used to solve the multicriteria problem. The flow process of the multicriteria reinforced Cuckoo search optimization is illustrated in Figure 2 [28]. Figure 2 demonstrates the flow process of the multicriteria reinforced Cuckoo search optimization for obtaining the optimal test case selection. The multicriteria reinforced Cuckoo search optimization is a population-based search process in which the number of Cuckoos (i.e. test cases) is initialized in search space.

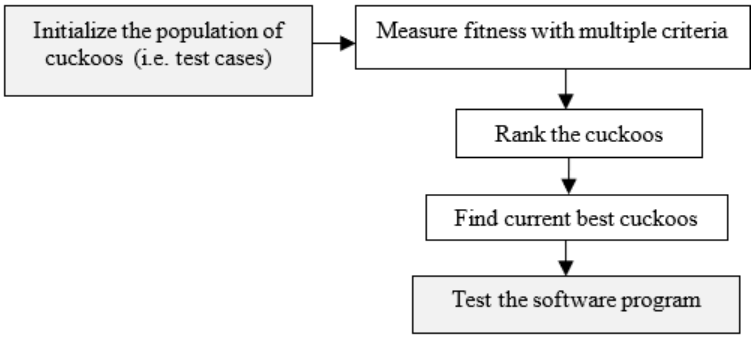

Figure 2. Flow process of multicriteria reinforced Cuckoo search optimization

In the search space, the population of the Cuckoo (i.e. test case) is initialized.

$$\delta_i \in \delta_1, \delta_2, \ldots . \delta_n \quad (1)$$

After the population generation, the fitness is computed based on the multicriteria function as (2).

$$\varphi_F = \arg\min(P_{\delta_i}, Q_{\delta_i}) \quad (2)$$

Where,φ_F denotes a fitness of each test case, 'P_{δ_i}' indicates the cost of the test case, 'Q_{δ_i}' symbolizes the time consumed by a test case to test the quality of a given software program. The cost of the test case 'P_{δ_i}' is measured as (3).

$$P_{\delta_i} = [S_\mu] + [T_\mu] + [K_\mu] \quad (3)$$

Where, 'S_μ' stand for memory utilized to store the source codes and 'T_μ' designates the amount of memory consumed to test the quality, 'K_μ' shows the memory utilized to store the operational knowledge of the software system. Then the time consumption to test the quality of a given software program is measured as (4).

$$Q_{\delta_i} = t\,[t_{SP}] \quad (4)$$

Where '$t(t_{SP})$' symbolizes the amount of time consumed by the test case to test the overall software product quality. Based on the fitness value, the test cases are ranked in an ascending order to find the current best solution.

From Table 1, the five test cases are ranked based on the fitness value. As a result, higher-ranked test cases are chosen to test the software product lines to predict the defects in each component. The algorithmic process of the multicriteria reinforced Cuckoo search optimization based test case selection is described as Algorithm 1.

Algorithm one given illustrates multicriteria reinforced Cuckoo search optimization to select the optimal test case for software code testing. Algorithm 1 shows the optimal test case selection to test the quality of given software programs taken from the dataset. The numbers of test cases are initialized in search space. After the initialization, the fitness of each test case is measured based on the multicriteria function. Based on the fitness value, the test cases ranked and find the current best solution. This process is iterated until the maximum iteration gets reached. Therefore, the selected test cases are used to test the quality of the program with minimum time.

Table 1. Test case ranking

Test Cases	Rank
$\boldsymbol{\delta_1}$	1
$\boldsymbol{\delta_2}$	2
$\boldsymbol{\delta_3}$	3
$\boldsymbol{\delta_4}$	4
$\boldsymbol{\delta_5}$	5

Algorithm 1: multicriteria reinforced Cuckoo search optimization

Input: Set of test cases in the test suite $\delta_1, \delta_2, \ldots . \delta_n$
Output: Select optimal test cases for software code testing
Begin
1. Initialize the population of test cases $\delta_1, \delta_2, \ldots . \delta_n$
2. While (t <max_iter)
3. For each test case δ_i
4. Compute the fitness 'φ_F'
5. Rank $\delta_1, \delta_2, \ldots . \delta_n$ based on φ_F
6. Find current best $\delta_1, \delta_2, \ldots . \delta_n$
7. Choose the current best $\delta_1, \delta_2, \ldots . \delta_n$ for testing the software program
8. End for
9. End while
10. End

3.2. Generative likelihood latent Dirichlet allocation model

After finding the optimal test cases, the generative latent Dirichlet allocation model is designed in the RCSOLDA-RIR technique for increasing the reliability of the system by predicting the software failure density based on the posterior probability distribution. Here, the generative model is a statistical model of the joint probability distribution on observable variables i.e. topics in software products, and a target variable i.e. failure occurrences.

In existing, LDA is unsupervised machine learning. It is a statistical topic model which has been broadly applied to abstract semantic information from software source code and a failure density by using mapping failures. LDA is used to predict the component failures for software source code topic mining. By applying the generative latent Dirichlet allocation model, the probability of the failure occurrences of the components is measured using optimal test cases. Therefore, the probability distribution [29] is obtained as (5).

$$P(\theta_{mc}, z_t \mid \omega_s, \alpha_p, \beta_f) = \frac{P(\theta_{mc}, z_t, \omega_s \mid \alpha_p, \beta_f)}{P(\omega_s \mid \alpha_p, \beta_f)} \quad (5)$$

Where, P denotes a probability of the failure occurrence, α_p is the parameter of the topic before distribution, β_f denotes a parameter of the failure distribution probability, θ_{mc} denotes a joint distribution of the topic mixture, z_t indicates topics in a software product, ω_s indicates a set of 'n' source code lines. The probability values are lie between 0 to 1. If the probability is higher, then the results confirmed that the maximum possibility of failure occurrences.

To improve the reliability of the system, the Software failure density [29] for each component is measured as (6).

$$SFD_c = \frac{Numberofsoftwarefailuresdetected}{numberoflinesofsoftwarecodes} \quad (6)$$

Where, SFD_c indicates a software failure density of components. The failure density describes that the failure occurred in the component at the time 't'. Using the above ratio as motivation, the failure density of a topic [29] is estimated as (7).

$$FD_t = \sum_{j=1}^{m} SFD_c(j) \quad (7)$$

Where, FD_t denotes a failure density of topics,$SFD_c(j)$ indicates a software failure density of the 'j's component. Using 7th equation, failures are mapped to topics for finding the failure-proneness of the topic. In this way, the failure density of all the topics in the software products is measured for accurate failure prediction of the software program.

Algorithm two process describes the step by step process of generative latent Dirichlet allocation for finding the software failure probability of each component and topics. With the chosen optimal test case, each line of source code topic for the given software product is tested and identified the failure density. This helps to design an efficient new software product according to the user needs. Generative latent Dirichlet allocation is used to reduce the service provisioning time.

Algorithm 2: Generative latent Dirichlet allocation

Input: Schoolmate Dataset, input software products$\gamma_1, \gamma_2, \gamma_3, ., \gamma_n$, topics $z_t = z_1 z_2, z_, \ldots z_t$,
Components $C_m = C_1 C_2, C_2, \ldots C_m$
Output: failure density prediction

```
Begin
    1.   For each topic z_t
    2.  For each Component 'C_m'
    3.                  Measure failure probability P(θ_mc, z_t | ω_s, α_p, β_f)with test cases
    4.                  Compute the software failure density 'SFD_c'
    5.                  Compute failure density of a topic  'FD_t'
    6.  End for
    7.   End for
End
```

3.3. Ruzicka indexive regression-based failure prediction

The proposed RCSOLDA-RIR technique uses the Ruzicka indexing regression for handling the large size of open-source software applications in the software quality management process to predict the component failure in the new version of the software products. Based on this concept, the technique provides better performance for analyzing the topics of consecutive versions of software products with a minimal amount of time consumption. Regression analysis is a set of statistical processes for measuring the relationships between two variables (i.e. consecutive versions of software products). The relationship between the consecutive versions of software products is measured using the Ruzicka similarity index. Regression is carried to test the system efficiently by using a test suite that focuses on important and very perceptible functionality.

Figure 3 illustrates the Ruzicka Similarity indexed regression for predicting the failure component in software products before its release. Let us consider the topics of consecutive versions of input software products 'z_i, z_{i+1} for software quality management. Jaccard's similarity coefficient is called Ruzicka Similarity. The Ruzicka Similarity index is measured based on the consecutive versions of software products for designing a new software product with the failure-free operation of the input software program. The Ruzicka similarity is expressed as (8).

$$\Phi = \frac{z_i \cap z_{i+1}}{\sum z_i + \sum z_{i+1} - z_i \cap z_{i+1}} \tag{8}$$

Where Φ symbolizes Ruzicka Similarity coefficient, z_i signifies the one version of the topic in a software product, z_{i+1}designates new version of topics in a similar software product, $z_i \cap z_{i+1}$ is a mutual dependence between the topics in a software product, $\sum z_i$ is the sum of z_i score, $\sum z_{i+1}$ is the sum of z_{i+1} score. Ruzicka similarity is used to provide better results between the similarity measures. The Ruzicka Similarity coefficient returns the output ranges from 0 to +1. The regression returns the similarity results by setting the threshold value as (9).

$$Y = \begin{cases} \Phi > \tau \,; highsimilarity \\ \Phi < \tau \,; Lesssimilarity \end{cases} \tag{9}$$

Where Y denotes a regression output, τ indicates the threshold. If the similarity value 'Φ' is higher than the threshold, then it is said to be higher similar. Otherwise, it returns less similarity. In this way, the similarities of the topics in the two neighboring versions are determined. When the failure component in the preceding version is presented in the new version hence it also creates failure while running the software program. In this case, these failures in the software products are resolved before the release to obtain the failure-free operation resulting in it increases reliability. As a result, the amount of time was minimized for providing the user satisfied software program based on their requirements.

Algorithm three given illustrates the Ruzicka indexing regression to obtain the failure-free software product for improving the software reliability. For each topic and component of the current version and the next consecutive version of the software product, measure the similarity. If the same topics are presented in the new version of the software product, then the failure component is predicted. Finally, the predicted failure components are corrected and obtain the reliability of the software product.

Algorithm 3: Ruzicka Similarity indexed regression-based failure prediction

```
Input: Schoolmate Dataset,  input software productsγ_1,γ_2,γ_3,..,γ_n, topics z_t = z_1,z_2,z_,…z_t  ,
Output: Improve software reliability
Begin
  1.  For each topic in the current version  'z_i'
  2.     For each topic in the new version 'z_i+1'
  3.            Measure Ruzicka Similarity coefficient 'Φ'
  4.             If (Φ > τ) then
  5.  Y   returns the high similarity
  6.                 else
```

```
  7.  Y   returns the less similarity
  8.            End if
  9.  Predict the failure component of the new version
  10. Resolve the failure component
  11. Obtain failure-free software product
  12. End for
  13. End for
End
```

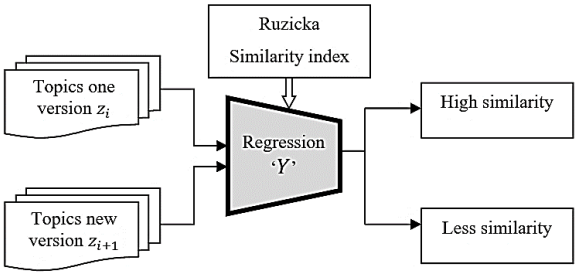

Figure 3. Ruzicka similarity indexed regression

4. EXPERIMENTAL SETTINGS

In this section, the RCSOLDA-RIR technique and existing methods namely WPA-PSO [1], and FPSONNM [2] are implemented in Java Language using the schoolmate data set. The SchoolMate dataset is taken from [30]. The dataset comprises the number of open-source PHP programs to perform software quality tests. For the experimental evaluation, 10-100 the size of software program source codes is considered. The hardware specifications are shown in Table 2.

Table 2. Hardware specifications

Hardware	Specification
Operating system	Windows 10
Processor	core i3-4130 3.40GHZ
RAM	4GB RAM
Hard disk	1TB (1000 GB)
Motherboard	ASUSTek P5G41C-M
Protocol	Internet

5. RESULT AND DISCUSSIONS

In this section, the comparative result analysis of the RCSOLDA-RIR technique and existing methods namely WPA-PSO [1], and FPSONNM [2] is discussed. The efficiency of the RCSOLDA-RIR technique is determined along with the metrics such as scalability, service provisioning time, and reliability with the help of tables and graphical representations.

5.1. Reliability

The reliability is measured as the ratio of the number of source code lines that are executed without any error to the total number of source code lines taken as input. It defines the failure-free operation of the input software program. The reliability is measured as (10).

$$R = \left(\frac{Z_{SE}}{n}\right) * 100 \quad (10)$$

Where, R denotes reliability,Z_{SE}' represent the number of source code lines that are successfully executed without any error, 'n' denotes the size of software program code taken as input. The reliability is calculated in terms of percentage (%). The experimental results of reliability are reported in Table 3.

Table 3 portrays the reliability of three techniques namely the RCSOLDA-RIR technique, WPA-PSO [1], and FPSONNM [2]. This is owing to the application of the Ruzicka similarity used in the RCSOLDA-RIR technique. Followed by, the quality of the software program gets improved and also increases reliability. The average comparison results demonstrate that the reliability of the proposed

RCSOLDA-RIR technique is considerably improved by 8% as compared to WPA-PSO [1], and 12% FPSONNM [2] respectively.

Table 3. Tabulation of reliability

Size of Software Program Code (KB)	Reliability (%)		
	RCSOLDA-RIR	WPA-PSO	FPSONNM
10	92	88	84
20	90	84	80
30	93	88	84
40	92	87	85
50	94	88	86
60	96	87	85
70	94	87	84
80	95	85	83
90	96	87	85
100	97	88	84

5.2. Scalability

Scalability is the ability of different algorithms to handle different sizes of input program code while testing the software quality process. The quantitative analysis of scalability is computed as (11).

$$S = \left(\frac{Z_{CT}}{n}\right) * 100 \quad (11)$$

Where 'S' denotes scalability, Z_{CT} denotes the number of source lines of code that are correctly tested, 'indicates the size of software program code taken as input. The scalability is measured in terms of percentage (%). The graphical representation of the scalability is described in Figure 4.

Figure 4. graphical illustration of scalability concerning the size of the software program code. Figure 4. portrays the scalability of the three techniques namely the RCSOLDA-RIR technique, WPA-PSO [1], and FPSONNM [2]. This is due to the application of Multicriteria reinforced Cuckoo search optimization is applied in the RCSOLDA-RIR technique. This is the reason for achieving the higher scalability of the RCSOLDA-RIR technique. The average scalability results of the RCSOLDA-RIR technique are increased by 6% when compared to WPA-PSO [1], and 11% when compared to FPSONNM [2].

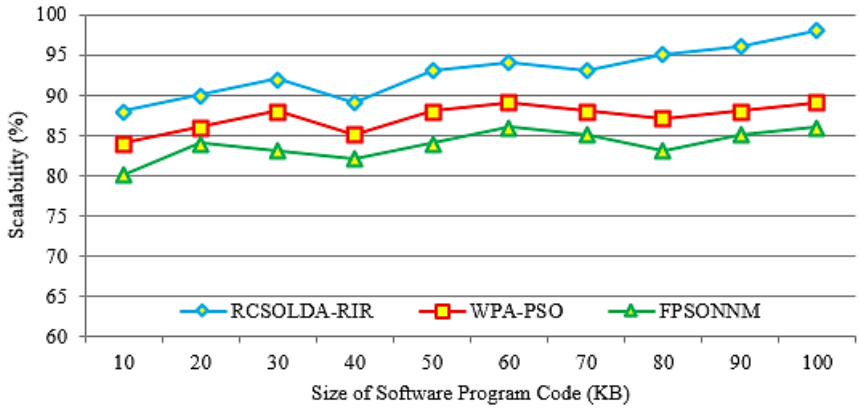

Figure 4. Graphical illustration of scalability

5.3. Service provisioning time

The service provisioning time measures the amount of time consumed by the algorithm for providing user satisfied services based on their requirements and estimated as (12).

$$SPT = [n] * time(CBSP) \quad (12)$$

Where 'SPT' indicates a service provisioning time of software program, 'denotes the size of software program code, '$time(CBSP)$'symbolizes the time taken for designing the best software product based on

user requirements. The provisioning time is measured in terms of milliseconds (ms). The service provisioning time of different techniques is reported in Table 4.

The experimental results of service provisioning time using three methods are illustrated in Table 4. The result is evidence that the proposed RCSOLDA-RIR technique uses the GLDA model achieves lesser service provisioning time when compared to existing techniques. The average results of the RCSOLDA-RIR technique are reduced the service provisioning time by 20% as compared to WPA-PSO [1], and 29% as compared to FPSONNM [2].

Table 4 Tabulation of Service provisioning time

Size of Software Program Code (KB)	Service provisioning time (ms)		
	RCSOLDA-RIR	WPA-PSO	FPSONNM
10	8	12	15
20	10	15	18
30	12	16	19
40	15	18	20
50	17	20	23
60	20	24	27
70	22	27	29
80	25	30	32
90	28	33	35
100	32	37	40

6. CONCLUSION

In this paper, the major objective of the proposed work is to get better software quality prediction by improving the reliability of software products. The multicriteria optimization technique is employed to test the software program quality by selecting the optimal test cases with higher scalability. The generative latent Dirichlet allocation model is applied to find the failure probability of each component for reducing the service provision time. The regression is applied to predict the component failure of a new version of the software product for achieving better reliability. The proposed work is compared with the two existing methods (i.e. WPA-PSO and FPSONNM). The results of the RCSOLDA-RIR technique provide better performance with an improvement of reliability and scalability by 10% and 9% and reduction of service provision time by 25% as compared to existing works. The proposed RCSOLDA-RIR technique achieves better scalability and reliability. The proposed work is further suggested to use new research work for correcting the failure of the software by using fault tolerance techniques.

REFERENCES

[1] L. Zhen,Y. Liu,W. Dongsheng, and Z. Wei, "Parameter Estimation of Software Reliability Model and Prediction Based on Hybrid Wolf Pack Algorithm and Particle Swarm Optimization," *IEEE Access*, vol. 8, pp. 29354-29369, 2020, doi: 10.1109/ACCESS.2020.2972826.

[2] P. Roy, G. S. Mahapatra, and K. N. Dey, "Forecasting of software reliability using neighborhood fuzzy particle swarm optimization based novel neural network," in *IEEE/CAA Journal of Automatica Sinica*, vol. 6, no. 6, pp. 1365-1383, November 2019, doi: 10.1109/JAS.2019.1911753.

[3] N. Guptaa, A. Sharmaa, and M. K. Pachariya, "Multi-objective test suite optimization for detection and localization of software faults," *Journal of King Saud University - Computer and Information Sciences*, pp. 1-13, 2020, doi: 10.1016/j.jksuci.2020.01.009.

[4] S. A. Sari and K. M. Mohamad, "Recent Research in Finding Optimal Path by Ant Colony Optimization," *Bulletin of Electrical Engineering and Informatics*, vol. 10, no. 2, pp. 1015-1023, 2020, doi: 10.11591/eei.v10i2.2690.

[5] M. A. A. K. Alabajee, N. A. A. Saati, and T. R. Alreffaee, "Parameter Tuning of Software Effort Estimation Models Using Antlion Optimization," *TELKOMNIKA* (*Telecommunication Computing Electronics and Control)*, vol. 19, no. 3, pp. 817-828, 2021, doi: 10.12928/telkomnika.v19i3.16907.

[6] G. Jabeen, P. Luo, and W. Afzal, "An improved software reliability prediction model by using high precision error iterative analysis method," *Software Testing, Verification and Reliability*, vol. 29, no. 6-7, 2019, doi: 10.1002/stvr.1710.

[7] G. Jabeen, P. Luo, and W. Afzal, "An improved software reliability prediction model by using high precision error iterative analysis method," *Software Testing, Verification and Reliability*, vol. 29, no. 6-7, pp. 1-22, 2019, doi: 10.1002/stvr.1710.

[8] Z. Li, M. Yu, D. Wang and H. Wei, "Using Hybrid Algorithm to Estimate and Predicate Based on Software Reliability Model," in *IEEE Access*, vol. 7, pp. 84268-84283, 2019, doi: 10.1109/ACCESS.2019.2917828.

[9] C. Diwaker *et al*., "A New Model for Predicting Component-Based Software Reliability Using Soft Computing," in *IEEE Access*, vol. 7, pp. 147191-147203, 2019, doi: 10.1109/ACCESS.2019.2946862.

[10] Q. Li and H. Pham, "A Generalized Software Reliability Growth Model With Consideration of the Uncertainty of Operating Environments," *IEEE Access*, vol. 7, pp. 84253-84267, 2019, doi: 10.1109/ACCESS.2019.2924084.
[11] K. Wang, T. T. Wang, and X. H. Su, 'Test case selection using multi-criteria optimization for effective fault localization," *Computing*, vol. 100, no. 8, pp. 787-808, 2018, doi: 10.1007/s00607-018-0610-0.
[12] N. Baskar and C Chandrasekar, "Optimized neuro-PSO-based software maintainability prediction using relief features selection method," *Indonesian Journal of Electrical Engineering and Computer Science (IJEECS)*, vol. 15, no. 3, pp. 1517-1526, 2019, doi: 10.11591/ijeecs.v15.i3.pp1517-1526.
[13] B. Mohammed, I. Awan, H. Ugail, and M. Younas, "Failure prediction using machine learning in a virtualised HPC system and application," *Cluster Computing*, vol. 22, pp. 471-485, 2019, doi: 10.1007/s10586-019-02917-1.
[14] C. Diwaker, P. Tomar, R. C. Poonia, and V. Singh, "Prediction of Software Reliability using Bio Inspired Soft Computing Techniques," *Journal of Medical Systems*, vol. 42, no. 5, pp. 1-16, 10 April, 2018, doi: 10.1007/s10916-018-0952-3.
[15] M. Cinque, D. Cotroneo, R. D. Corte, and A. Pecchia, "A framework for on-line timing error detection in software systems," *Future Generation Computer Systems*, vol. 90, pp. 521-538, 2019, doi: 10.1016/j.future.2018.08.025.
[16] Z. Li, X. Y. Jing, F. Wu, X. Zhu, B. Xu, and S. Ying, "Cost-sensitive transfer kernel canonical correlation analysis for heterogeneous defect prediction," *Automated Software Engineering*, vol. 25, no. 1, pp. 1-45, 2018, doi: 10.1007/s10515-017-0220-7.
[17] A. Marjuni, T. B. Adji, and R. Ferdiana, "Unsupervised software defect prediction using median absolute deviation threshold based spectral classifier on signed Laplacian matrix," *Journal of Big Data*, vol. 6, pp. 1-20, 2019, doi: 10.1186/s40537-019-0250-z.
[18] Z. W. Zhang, X. Y. Jing, and T. Wang, "Label propagation based semi-supervised learning for software defect prediction," *Automated Software Engineering*, vol. 24, pp. 47-69, 2017, doi: 10.1007/s10515-016-0194-x.
[19] Z. Xu *et al.*, "LDFR: Learning deep feature representation for software defect prediction," *Journal of Systems and Software*, vol. 158, pp. 1-20, 2019, doi: 10.1016/j.jss.2019.110402.
[20] G. Fan, X. Diao, H. Yu, K. Yang, and L. Chen, "Software Defect Prediction via Attention-Based Recurrent Neural Network," *Scientific Programming*, vol. 2019, pp. 1-14, 2019, doi: 10.1155/2019/6230953.
[21] J. Wang and C. Zhang, "Software Reliability Prediction Using a Deep Learning Model based on the RNN Encoder-Decoder," *Reliability Engineering & System Safety*, vol. 170, pp. 73-82, 2018, doi: 10.1016/j.ress.2017.10.019.
[22] H. Bousqaoui, I. Slimani, and S. Achchab, "Comparative analysis of short-term demand predicting models using ARIMA and deep learning," *International Journal of Electrical and Computer Engineering*, vol. 11, no. 4, pp. 3319-3328, 2019, doi: 10.11591/ijece.v11i4.pp3319-3328.
[23] Z. Rustam, F. Zhafarina, G. S. Saragih, and S. Hartini "Pancreatic cancer classification using logistic regression and random forest," *IAES International Journal of Artificial Intelligence (IJ-AI)*, vol. 10, no. 1, pp. 476-481, 2021, doi: 10.11591/ijai.v10.i2.pp476-481.
[24] M. G. Valls, J. E. Barreno, and J. G. Muñoz, "An extensible collaborative framework for monitoring software quality in critical systems," *Information and Software Technology*, vol. 107, pp. 3-17, 2019, doi: 10.1016/j.infsof.2018.10.005.
[25] S. Huda *et al.*, "An Ensemble Oversampling Model for Class Imbalance Problem in Software Defect Prediction," in *IEEE Access*, vol. 6, pp. 24184-24195, 2018, doi: 10.1109/ACCESS.2018.2817572.
[26] E. A. Felix and S. P. Lee, "Predicting the number of defects in a new software version," *PLOS ONE*, vol. 15, no. 3, pp. 1-30, March 2020, doi: 10.1371/journal.pone.0229131.
[27] A. M. Hassan and A. John, "Comparative analysis on different software piracy prevention techniques," *International Journal of Informatics and Communication Technology (IJ-ICT)*, vol. 10, no. 1, pp. 1-8, 2015, doi: 10.11591/ijict.v10i1.pp1-8.
[28] K. S. Kumar and A. M. Kumaravel, "Optimal Test Suite Selection using Improved Cuckoo Search Algorithm Based on Extensive Testing Constraints," *International Journal of Applied Engineering Research*, vol. 12, no. 9, pp. 1920-1928, 2017, [Online]. Avalable: https://www.ripublication.com/ijaer17/ijaerv12n9_22.pdf
[29] H. Liu, L. Xu, M. Yang, M. Yan, and Z. Zhang, "Predicting Component Failures Using Latent Dirichlet Allocation," *Mathematical Problems in Engineering*, vol. 2015, vol. 6, pp. 1-15, doi: 10.1155/2015/562716.
[30] SchoolMate Dataset, 2013. [Online]. Available: https://sourceforge.net/projects/schoolmate/?source=directory