Software Quality Management for open Source Software Based on Louvain Parallelization Heuristic and Greedy Discritized Optimization

R. Chennappan[a],Dr. Vidyaa Thulasiraman[b]

[a]Research Scholar, Department of Computer Science, Periyar university, Salem- 636011, Tamilnadu, India.
[b]Assistant Professor & Head, Department of Computer Science, Government Arts College for Women, Bargur- 635104, Tamilnadu, India
E-mail: chennappanphd@gmail.com

Article History:*Do not touch during review process(xxxx)*

Abstract: Designing reliable software product is becoming more difficult as software becomes ubiquitous and is deployed on software quality management. Software quality detection has become the fundamental operation. Any changes made in the source lines of codes of software products has an adverse effect on the long run and therefore compromising the scalability and reliability of software product users. Few research works have been introduced in existing work for software quality prediction using various data mining techniques. But, scalability and reliability of software quality management was not enough. Besides to that, the time required by the conventional research work for performing the service provisioning was too high. For that reason, the proposed research work is concentrated to address these issues by providing the higher scalability and reliability with minimum amount of time consumption for service provisioning during the software quality management process. Thus, the research work introduces the proposed Louvain Parallelization Heuristic Based Greedy Discritized Optimization (LPH-GDO) Model for performing fast software quality prediction in a significant manner. In this present work, the experimental evaluation of LPH-GDO Model has been conducted on metrics such as scalability, service provisioning time and software reliability with respect to different size of software program code.

Keywords: Absolute Importance Rating, Modularity, Objective Function, Reliability, Software Product, Source Lines of Code, Test Cases

1. Introduction

In general, one of the key aspects of software development is how to make predictions and assessments of quality and reliability for developed products. Software quality prediction is one of the challenges process in software engineering fieldowing to lack of sufficient tools to evaluate software codes. Predictingdefects in software products is a difficult process. Mainly, when the sizes of software products grow, Software quality prediction becomes costly with complicated testing and evaluation mechanisms.

The neural network based software quality prediction (NN-SQP) technique was implemented in [1] for predicting the quality of the resulting software. However, scalability of software quality management process was not considered. Advanced neural network and Hybrid Cuckoo search (HCS) optimization algorithm was applied in [2] for increasingdetection accuracy of software quality.Here, the HCS was utilized for enhancing the neural network that results in the optimization of weight factor to increase the prediction accuracy. However, reliability was lower.

Bounded Generalized Pareto distribution (BGPD) model was designed in [3] to analyze the fault of open source software and thereby improvingthe software development. However, service provisioning time was more while considering the large size of software program codes. A novel approach was introduced in [4] to increase the performance of software reliability detectionusing time series modeling. But, scalability using this method was remained open issue.

A decision tree model was employed in [5] for determining reliability of a software program and thereby achieving higher software quality. However, time taken for getting g higher softwarereliability was more. In [6], the Linear Regression Models was implemented for performing better software development process by the reduction of estimated cost for software products. But, testing time of this model was not reduced.

A wrapper based feature selection approach and an ensemble learning algorithm (RUSBoost) was employed in [7] for enhancing software quality detection.However, the amount of processing time needed for improving software quality was more. Perspective-based model was applied in [8] wherea property-attribute relationshipwas identified using Grounded Theory Analysis. However, detailed elicitation of these contextual factors that grows with coded portion was remained unaddressed during the analyzation of characteristics of quality for software.

The stochastic Markov chain framework was presented in [9] to get better software quality management and selection of optimum set of factors influencing software quality and therefore improving scalability. However, response time was not concentrated for performing better software quality management. An open source software tool for quality control in liquid chromatography-high resolution mass spectrometry (LC-HRMS) was constructed in [10]. But, service provisioning time using LC-HRMS was very higher.

A lot of research works have been designed for software quality prediction using different techniques. However, scalability and reliability of software quality management was not sufficient. In order to overcome these drawbacks, LPH-GDO Model is designed in this research work.

2. Related Works

Bio Inspired Soft Computing Techniques were developed in [11] for enhancing detection performance of software reliability. An adaptive software quality model extraction methodology was employed in [12] for software product quality assessment.

Machine learning techniques were introduced in [13] to discover the software quality in the early stages of software development.A novel technique was presented in [14] for attaining highersoftware applications quality by means ofconsidering the contribution relationship among quality attributes.

Ant colony optimization algorithm was applied in [15] for evaluating software quality at the early stages of the design process. A component probabilistic dependency graph (CPDG) was developed in [16] for early reliability prediction.

The impact of asynchrony change pattern on development and code smells such as anti-patterns and code clones was analyzed in [17]. Proactive Self-Adaptation was introduced in [18] for achieving enhanced software reliability.

Machine learning and meta-heuristics techniques designed for software optimization at compile-time and run-time was presented in [19]. A survey of different methods designed for increasing thesoftware reliability using software engineering was analyzed in [20].

- Even though lots of research works were developed for software quality management, providing of better results for enhancing the scalability and reliability of software quality management with large size of open source software application remains as a major issue. In order to resolve the existing issues related to obtain better scalability and reliability of software quality management, the research work introduces the novel proposed LPH-GDO Model with the combination of parallelization process in Louvain algorithm on contrary to conventional works.

The main contributions of REOF-BSTC Technique is formulated as,

- To improve the scalability and reliability of software products with less service provisioning time, the efficient proposed LPH-GDO Model is implemented. This is achieved in the proposed LPH-GDO Model with the implementation of two essential algorithms such as Louvain Parallelization Heuristic algorithm and Greedy Discritized Optimization algorithm.
- To increase the quality of open source software products, the proposed LPH-GDO Model executes the Louvain Parallelization Heuristic algorithm.
- During software quality management, the proposed LPH-GDO Model utilizes the parallelization to successfully handle the large size of open source software application.
- To provide better reliability for software program with a minimal test cost, the proposed LPH-GDO Model applies the Greedy Discritized Optimization algorithm after the designing of best software product.

3. Louvain Parallelization Heuristic Based Greedy Discritized Optimization Model

The Louvain Parallelization Heuristic Based Greedy Discritized Optimization (LPH-GDO) Model is introduced in order to increases the performance software quality detection with a minimal time complexityduring software quality management process. In the proposed LPH-GDO Model, the performance of software quality detection process is evaluated by considering the measurement of three feature metrics such as scalability, service provisioning time and reliability. On the contrary to existing works, LPH-GDO Model is designed where parallelization heuristics is employed for fast software quality predictionusing the Louvain method. In LPH-GDO Model, parallel implementation helps to gethigher quality software program outputs in a fewer iteration as compared to state-of-the-art works.The architecture diagram of LPH-GDO Model is demonstrated in below Figure1.

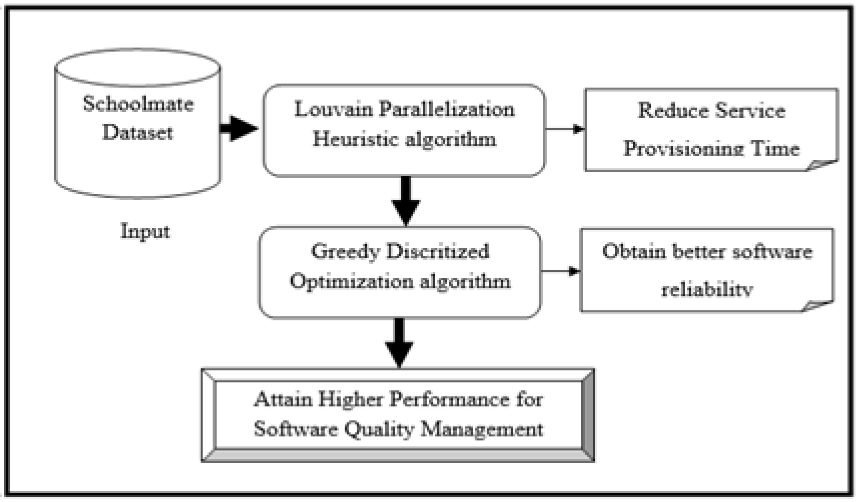

Schoolmate Dataset
Input
Louvain Parallelization Heuristic algorithm
Reduce Service Provisioning Time
Greedy Discritized Optimization algorithm
Obtain better software reliability
Attain Higher Performance for Software Quality Management

Figure 1 Architecture Diagram of LPH-GDO Model for Software Quality Management

Figure 1 presents the overall processes of LPH-GDO Model to get better software quality detection performance. As demonstrated in above figure, LPH-GDO Model at first takes schoolmate dataset as input. Next,LPH-GDO Model apply Louvain Parallelization Heuristic algorithm with objective of minimizing the service provisioning time of open software products. After that,LPH-GDO Model appliesGreedy Discritized Optimization algorithm with the intention of obtaining higher software reliability by performing failure-free operation of input software program. Thus, LPH-GDO Modelachievesimproved performance for software quality management as compared to state-of-the-art works. The exhaustivedetail about LPH-GDO Model is described in belowsubsections.

3.1 Louvain Parallelization Heuristic algorithm

In the LPH-GDO Model, the Louvain Parallelization Heuristic algorithm is introduced that combines theparallelization process in Louvain algorithm to increase the quality of open source software products by improving the scalability of software quality management process with a lower service provisioning time. In LPH-GDO Model, parallelizationhelps for handling the large size of open source software application during software quality management process. From that, LPH-GDO Model provides better performance for analyzing the quality of consecutive versions of software products with a minimal amount of time consumption. The process of Louvain Parallelization Heuristic algorithm is depicted in below Figure 2.

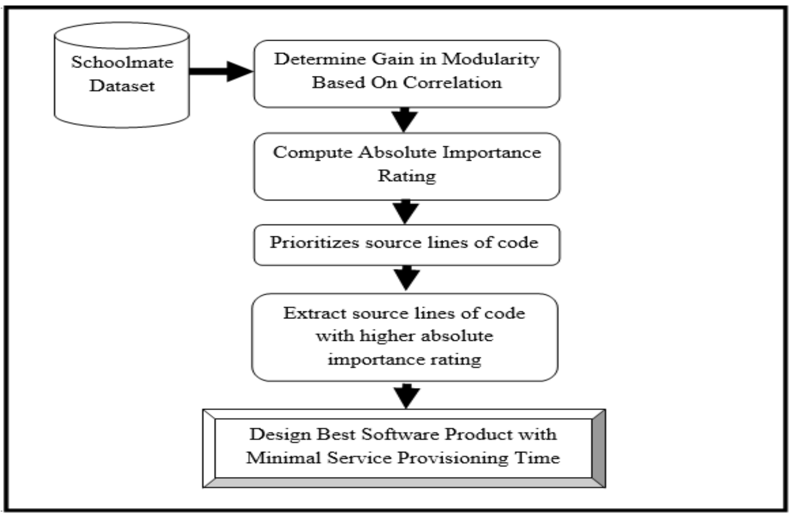

Schoolmate Dataset
Determine Gain in Modularity Based On Correlation
Compute Absolute Importance Rating
Prioritizes source lines of code
Extract source lines of code with higher absolute importance rating
Design Best Software Product with Minimal Service Provisioning Time

Figure 2 Block Diagram of Louvain Parallelization Heuristic algorithm

Figure 2 shows the flow process of Louvain Parallelization Heuristic algorithm to attain better software quality management with a lower service provisioning time. TheLouvain Parallelization Heuristic algorithm isanagglomerativealgorithmthatstartswitheachnodeis assignedto unique open source programs (i.e. consecutive versions of software products). Here, each node represents source lines of codeofeach consecutive version. ThisLouvain Parallelization Heuristic algorithm performsmultiplepassesuntilthebestopen source product isdesigned based on the user requirement.Let us consider the consecutive versions of input software product are represented as '$\gamma_1, \gamma_2, \gamma_3, ., \gamma_n$. Here, '$\gamma_i$' represents the different version of software product (source lines of code) considered for software quality management. InLouvain Parallelization Heuristic algorithm,each pass contains two mainprocesses. During thefirstprocess,Louvain Parallelization Heuristic algorithm measures thegaininmodularitybased oncorrelation between the two consecutive versions of software products using the Tanimoto Correlation Coefficient. On the contrary to state-of-the-art works, Louvain Parallelization Heuristic algorithm employs Tanimoto Correlation Coefficient in order to identify the correlation between the consecutive versions of software products and thereby designing the new software product according to user needs.

The correlation between the consecutive versions 'γ_i'and 'γ_j'is determined using the following equations,

$$\rho(\gamma_i, \gamma_j) = \frac{n * \sum \gamma_i \gamma_j}{\sum {\gamma_i}_1{}^2 + \sum {\gamma_j}_2{}^2 - \sum \gamma_i \gamma_j} (1)$$

From equation (1),'ρ' denotes a tanimoto correlation coefficient, 'n' represents the number of consecutive versions of input program, 'γ_i, γ_j' denotes a two consecutive versions and$\sum \gamma_i{}^2$ denotes a sum of squared score of the version 'γ_i'. Here,$\sum \gamma_j{}^2$denotes a sum of squared score of the version 'γ_j'. Here,'$\sum \gamma_i \gamma_j$' represents the sum of the product of the paired score of 'γ_i' and 'γ_j. The correlation coefficient provides the output results from 0 to +1. Here, '+1' indicates the high similarity between the twoconsecutive versionand '0' represents the low similarity between the two consecutive version.

Based on the correlation measurement, absolute importance rating 'ω' is measured for each version of software product using below mathematical representation,

$$\omega \rightarrow [\gamma_1, \gamma_2, \gamma_3, ., \gamma_n] \qquad (2)$$

From the above equation (2), 'ω' denotes the absolute importance rating is assigned for current version to the different newer versions of open source software products.The source lines of code with high absolute importance rating are more related to user requirements.During the second process, LPH-GDO Modelprioritizesthe eachopen source software product according to absolute importance rating. From that, source lines of code with high absolute importance rating are selected forproducing best software products using below mathematical representation,$\gamma_i^* = \arg\max\ \omega\ [\gamma_1, \gamma_2, \gamma_3, ., \gamma_n]$ (3)

From equation (3),LPH-GDO Model extractsthe source lines of code with maximum absolute importance ratings to develophigher quality open source software application. TheLouvain Parallelization Heuristic algorithm isveryfastduetothefactthatthenumberofsource lines of code taken for analyzing software quality managementis reduceddrasticallyafterthefirstfewpasses. Therefore, Louvain Parallelization Heuristic algorithm minimizesamountofcomputations.In addition,thecorrelation between conventional and contemporary version areveryeasytocalculatewiththegivenformula.

The algorithmic process of Louvain Parallelization Heuristic algorithm is explained in below,

```
// **Louvain Parallelization Heuristic algorithm**
Input:Schoolmate Dataset
Output:Achieve Higher Scalability and Minimal Service Provisioning Time
Step 1: Begin
Step 2:     For each open source software product '$\gamma_i$'
Step 3:         For each software product lines
Step 4:Measure gain in modularitybased oncorrelation between consecutive versions
'$\rho\ (\gamma_i, \gamma_j)$' using (1)
Step 5:Calculate absolute importance rating '$\omega$' using (2)
Step 6:Prioritizessource lines of code based on '$\omega$'
Step 7: Choose source lines of code using (3)
Step 8: Develop best software products according to user needs
Step 9:         End For
Step 10:    End For
Step 11: End
```

Algorithm 1 Louvain Parallelization Heuristic algorithm

Algorithm 1 step by step processes of Louvain Parallelization Heuristic algorithm. By using the above algorithmic steps, the Louvain Parallelization Heuristic algorithm initially get schoolmate dataset as input. After that, the Louvain Parallelization Heuristic algorithm calculates gain in modularity by considering correlation between consecutive versions of software products. Next, theLouvain Parallelization Heuristic algorithm determinesabsolute importance rating for each source lines of code. Followed by, the Louvain Parallelization Heuristic algorithm prioritizes source lines of code depends on absolute importance rating. Then, theLouvain Parallelization Heuristic algorithm find outs the source lines of code with high absolute importance rating in order to design best software product with a minimal time complexity. By using the above algorithmic process, LPH-GDO Model effectively improves software quality management performance while increasing the sizes of input of source lines of code with a minimal time. Hence, LPH-GDO Model enhances the scalability of software quality management as compared to conventional works.

3.2 Greedy DiscritizedOptimizationalgorithm

After designing best software product, Greedy Discritized Optimization algorithm is developedin LPH-GDO model with objective of increasing the reliability of software program with a minimal test cost. The Greedy Discritized Optimization algorithm identifies the optimal test suites in terms oftime and cost to get higher open software product quality. The Greedy Discritized Optimization algorithm initially takes number of test suites as input. Each test suite contains number of test cases. Then, Greedy Discritized Optimization algorithm defines the objective function. Here, objective function is to select the test suites with minimum test time and test cost for enhancing the reliability and quality of software products. The Greedy Discritized Optimization algorithm chooses the test suites which satisfy the objective function. By using the optimal test suites, Greedy Discritized Optimization algorithm achieves better software reliability and also reduces the testing cost of software program.

Let us consider number of test suites '$\varphi_i = \varphi_1, \varphi_2, .., \varphi_N$' with different number of test cases 'δ_i'. In Greedy Discritized Optimization algorithm,objective function 'OF' is mathematically defined using below,

$$OF = \arg\min [P_{\delta_i}, Q_{\delta_i}] \quad (4)$$

From the above equation (4), 'P_{δ_i}' denotes the cost of test case 'δ_i' in test suite 'φ_i' whereas 'Q_{δ_i}' represents time utilizedby test case for testing the software product quality. The costdetermines the cost involvedwhen sharing information between users that contains source code, test cases and operational knowledge. In Greedy Discritized Optimization algorithm, cost is calculated in terms of amount of memory employedto store the source code, test cases and operational knowledge. Hence, the cost of test case'P_{δ_i}' is determined using below,

$$P_{\delta_i} = S_\mu + T_\mu + K_\mu (5)$$

From the equation (5), 'S_μ'represents the amount of memory usedto store the codes of given open source software product and 'T_μ'indicates the amount of memory utilized for test cases to test the quality. Here,'K_μ'point outs the memory taken to store the operational knowledge of software system. Followed by, testing time taken by test case 'Q_{δ_i}'is mathematically determined using below,

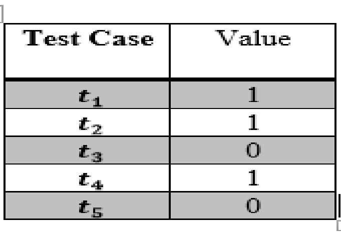
$$Q_{\delta_i} = t(t_{SP})(6)$$

From the equation (6), '$t(t_{SP})$'represent the amount of time requiredby test casefor testing the overall software product quality. Consequently, Greedy Discritized Optimization algorithm constructs a matrix called as TR table based on objective function by using all test cases in test suites. The TR table value is mathematically obtained using below mathematical expression,

$$TR(i,j) = \begin{cases} 1 \\ 0 \end{cases} \quad (7)$$

From the above equation (7), TR table value is determined for each test case 'δ_i' in test suite. In the above equation, the TR table value is obtained as 1 when the δ_i satisfy the objective function. Otherwise, the TR table value is obtained as 0 if thet_i

Table 1 TR table



Test Case	Value
t_1	1
t_2	1
t_3	0
t_4	1
t_5	0

cannot satisfy the objective function. The example for TR table is depicted in below.

Based on value of TR table, Greedy Discritized Optimization algorithm selects the optimal test cases in test suites for testing the reliability of given open source software products. Let us consider five test suites '$\varphi_1, \varphi_2, \varphi_3, \varphi_4, \varphi_5$' with different test cases'$\varphi_1 = \{\delta_2, \delta_3, \delta_5, \delta_6, \delta_8, \delta_9\}, \varphi_2 = \{\delta_1, \delta_2, \delta_4, \delta_5\}, \varphi_3 = \{\delta_4, \delta_7\}, \varphi_4 =$

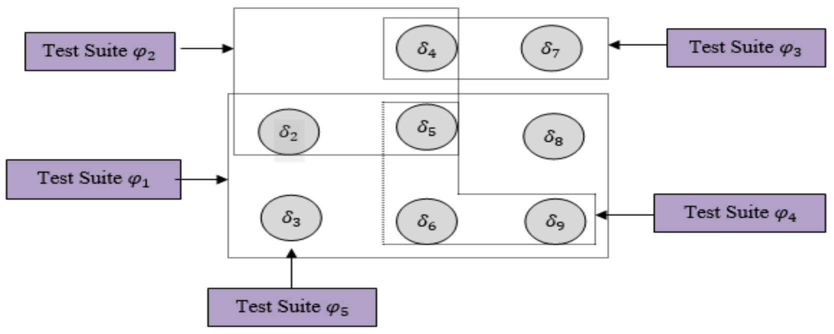


Figure 3 Greedy Discritized Optimization algorithm for Enhancing Software Reliability

$\{\delta_5, \delta_6, \delta_9\}, \varphi_5 = \{\delta_2, \delta_3\}$' as demonstrated in below Figure 3.

Figure 3 depicts the examples for Greedy Discritized Optimization algorithm. As illustrated in figure 5, the Greedy Discritized Optimization algorithm in LPH-GDO Model iteratively identifiestest cases which coverobjective functionsas optimal for performing software testing process with a minimal time and cost. From the above Figure 3, Greedy Discritized Optimization algorithm findsout the test suites 'φ_3, φ_4 ,φ_5' asoptimal to test the software quality. By using the chosen optimal test suites, then Greedy Discritized Optimization algorithm determines the probability of failures. The probability of failures is measured based on number of defects in input open source software productsusing below mathematical expression,

$$D_N = \frac{X}{Y} \quad (8)$$

From the equation (8), 'D_N' represent the number of defects that are identified in given software program whereas 'X'denotes number of test cases that are failure during testing process and 'Y' indicates number of test cases considered for testing the software reliability. By using the above mathematical formulation, Greedy Discritized Optimization algorithm finds the number of failures in a given open source software application.This helps for Greedy Discritized Optimization algorithm to increase the reliabilityof software products with a lower time and cost.

The algorithmic steps of Greedy Discritized Optimization algorithm isexplained in below,

```
// Greedy Discritized Optimization Algorithm
Input: Test Suites : $\varphi = \{\varphi_1, \varphi_2, \varphi_3, \varphi_4, \varphi_5\}$ and Schoolmate Dataset
Output: Enhance reliability of software products with minimal testing cost
Step 1: Begin
Step 2:    Define objective function '$OF$' using (4)
Step 3:    For each Test Suite '$\varphi_i$'
Step 4:        For each Test Case '$\delta_i$'
Step 5:        Measure testing cost '$P_{\delta_i}$' using (5)
Step 6:            Calculate testing time '$Q_{\delta_i}$' using (6)
Step 7:        End for
Step 8:    End for
Step 9:    Construct TR table using (7)
Step 10:   Select the test cases which covers objective function
Step 11:   Increase quality of software product by finding defects with '$\delta_i$'using (8)
Step 12:   Returnminimal cardinality subset of test suites
Step 13:End
```

Algorithm 2 Greedy Discritized Optimization

Algorithm 2 explains the step by step process of Greedy Discritized Optimization algorithm. With aid of the above algorithmic steps, Greedy Discritized Optimization algorithm initially describes the objective function. Subsequently, Greedy Discritized Optimization algorithm determines testing cost and time for each test case in test suites. Based on estimated testing cost and time, then Greedy Discritized Optimization algorithm creates the TR table. From that, Greedy Discritized Optimization algorithm discovers the optimal test cases for improving the quality. Finally, Greedy Discritized Optimization algorithm enhances thereliability of software product throughidentifying the number of defects using chosen optimal test cases. Thus, LPH-GDO Model attains better testing time and cost for improvingreliability of software products as compared to conventional works.

4. Experimental Settings

In order to evaluate the performance of proposed, LPH-GDO Model is implemented in Java Language using schoolmate dataset. The schoolmate dataset taken from https://sourceforge.net/projects/schoolmate/?source=directoryfor conducting experimental process contains many PHP open source software program. The LPH-GDO Model considers different size of software program code in the range of 10 KB-100 KB to perform experimental evaluation. Theeffectiveness of LPH-GDO Model is determined in terms of scalability, service provisioning time and reliability. The performance of LPH-GDO Model is compared with existing neural network based software quality prediction (NN-SQP) technique [1] and Advanced Neural network with Hybrid Cuckoo search(HCS) [2].

5. Result and Discussions

In this section, the comparative result analysis of LPH-GDO Model is discussed. The efficiency of LPH-GDO Model is determined along with the metrics such as scalability, service provisioning time and reliabilitywith the help of tables and graphical representations.

5.1 Performance Measure of Scalability

The scalability 'S'determines the ability of LPH-GDO Model to handle a large size of program code during the software quality management process. The scalability ismathematically measured using below,

$$S = \frac{Z_{CT}}{n} \qquad (9)$$

From equation (9), 'n'indicates the size of software program code taken as input and 'Z_{CT}'represents number of source lines of code that are correctly tested. The scalability is estimated in terms of percentage (%). When the scalability of software quality management is higher, the method is said to be more effective.

In order to measure the scalability of software quality prediction and management process, LPH-GDO Model is implemented in java language by considering various size of software program code in the range of 10 KB to 100 KB. When implementing the proposed LPH-GDO Model using 80 KB software program code size,proposed LPH-GDO Model gets 93 % scalability whereas conventional NN-SQP technique [1] and Advanced Neural network with HCS [2] obtains 74 % and 86 % respectively. Accordingly, scalability of software quality management using proposed LPH-GDO Model is very higher as compared to other existing works. The experimental result of scalability is presented in below Table 2.

Table 2 Tabulation for Scalability

Size of Software Program Code (KB)	Scalability (%)		
	NN-SQP technique	Advanced Neural network with HCS	LPH-GDO Model
10	66	75	85
20	69	77	87
30	67	80	89
40	68	78	86
50	71	81	88
60	69	82	90
70	72	84	91
80	74	86	93
90	76	83	95
100	80	88	97

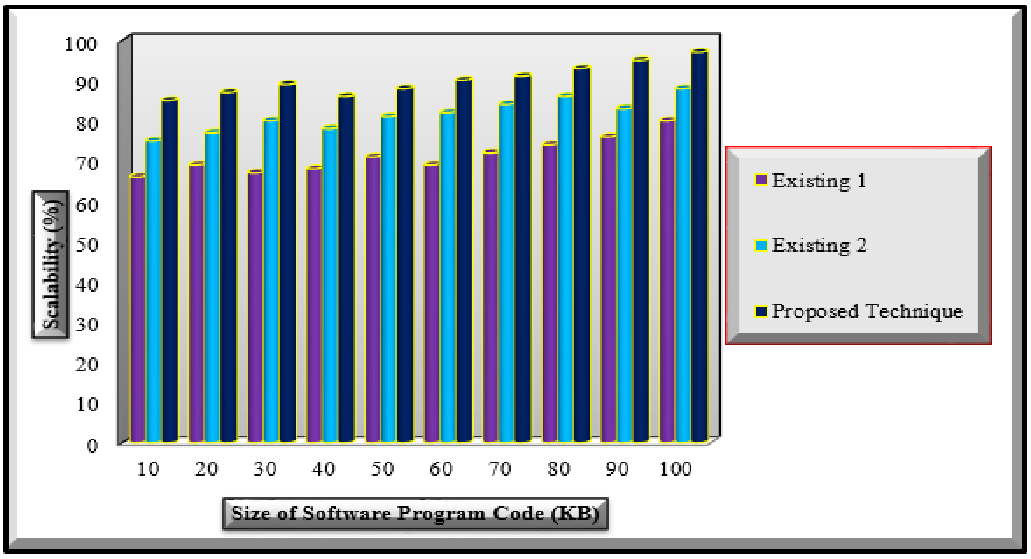


Figure 4 Comparative Result Analysis of Scalability versus Different Size of Software Program Code

Figure 4 shows the comparative result analysis of scalability with respect to different size of software program code from schoolmate dataset using three methods namely, NN-SQP technique [1], Advanced Neural network with HCS [2] and proposed LPH-GDO Model. As demonstrated in above figure, proposed LPH-GDO Model provides better scalability for attaining higher software reliability as compared to NN-SQP technique [1], Advanced Neural network with HCS [2] respectively. This is due to application of Louvain Parallelization Heuristic algorithm in proposed LPH-GDO where it computes gain in modularity depends on correlation between consecutive versions of software product lines. Subsequently, the Louvain Parallelization Heuristic Model calculates absolute importance rating toextract the source lines of code to create best software product with a lower time. From that, proposed LPH-GDO Model provides better software quality management performance while increasing the sizes of input of source lines of code. Therefore, proposed LPH-GDO Model improves scalability of software quality management by 27 % and 11 % when NN-SQP technique [1], Advanced Neural network with HCS [2] respectively.

5.2 Performance Measure of Service Provisioning Time

Service Provisioning Time (SPT) measures the amount of time required forproviding user satisfied software program based on their requirements. The service provisioning time is mathematically determined using below mathematical expression,

$$SPT = n * time(CBSP)(10)$$

From the equation (10), service provisioning time of software program is measured where 'n'denotes the size of software program code taken as input and '$time(CBSP)$'represents time needed for designing best software product based on user needs. When the service provisioning time is lower, the method is said to be more effectual. The provisioning time isevaluated in terms of milliseconds (ms).

The proposed LPH-GDO Model is implemented in java language with different size of software program code ranges from 10 KB to 100 KB in order to estimate the service provisioning time while performing the software quality management process. If the software program code size is taken as 50KB, then the proposed LPH-GDO Model requires 19ms to perform service provisioning as well as NN-SQP technique [1] and Advanced Neural network with HCS [2] consume 37 ms and 33 ms service provisioning time to provide user satisfied software program. From that, it is clear that, the proposed LPH-GDO Model requires less amount of time utilization for providing user satisfied software program based on their user requirements when compared to other conventional works. The comparative result of service provisioning time is depicted in below Table 3.

Table 3 Tabulation for Service Provisioning Time

Size of Software Program Code (KB)	Service Provisioning Time (ms)		
	Existing 1	Existing 2	Proposed Technique
10	23	16	9
20	26	22	11
30	31	26	14
40	34	30	16
50	37	33	19
60	40	39	22
70	45	42	26
80	51	47	28
90	55	53	31
100	61	59	36

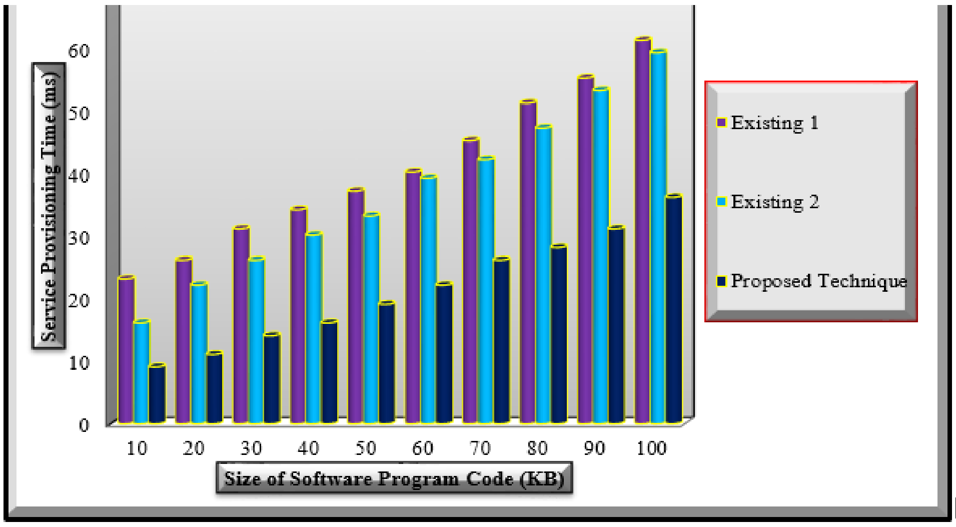


Figure 5 Comparative Result Analysis of Service Provisioning Time versus Different Size of Software Program Code

Figure 5 illustrates the experimental result analysis of service provisioning time based on diverse size of software program code from schoolmate dataset using three methods namely, NN-SQP technique [1], Advanced Neural network with HCS [2] and proposed LPH-GDO Model. As presented in above figure, proposed LPH-GDO Model provides better service provisioning time when compared to NN-SQP technique [1], Advanced Neural network with HCS [2] respectively. This is because of application of Louvain Parallelization Heuristic algorithm in proposed LPH-GDO model. By using the above algorithmic steps, proposed LPH-GDO model estimates gain in modularity using correlation between consecutive versions of software product lines. This helps for proposed LPH-GDO model to provide user satisfied software program based on their requirements with a minimal amount of time consumption. As a result, proposed LPH-GDO Model minimizesservice provisioning time by 49 % and 43 % as NN-SQP technique [1], Advanced Neural network with HCS [2] respectively.

5.3 Performance Measure of Reliability

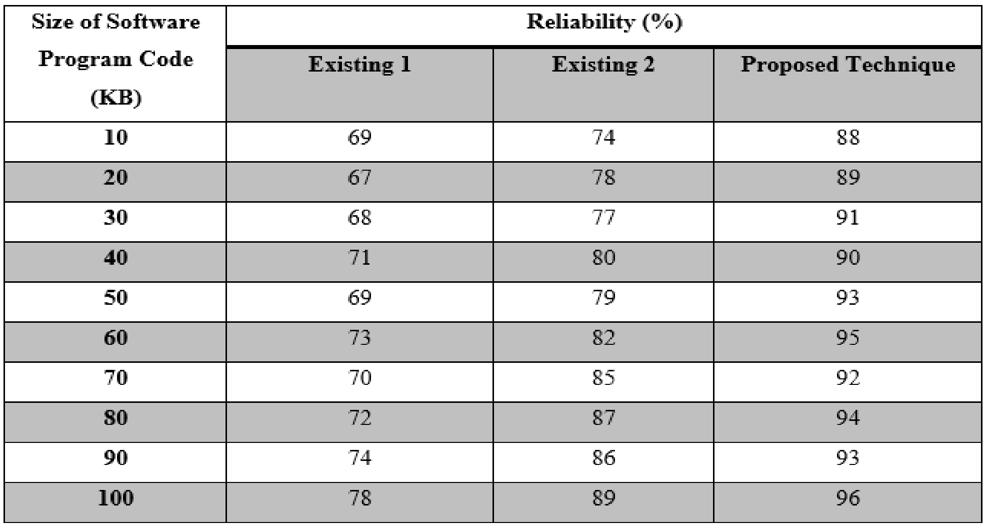
Reliability 'R'representsthe failure-free operation of input softwareprogram. From that, reliability is measured as the ratio of number of source code lines that are executed without any error to the total number of source code lines taken as input. The reliability is mathematically obtained using below,

$$R = \frac{Z_{SE}}{n} \qquad (11)$$

From the above equation (11), 'n' denotes the size of software program code taken as input whereas 'Z_{SE}' represent the number of source code lines that are successfully executedwithout any error. The reliability is calculated in terms of percentage (%). When the reliability is higher, the method is said to be more effective.

To estimate the reliability of software products during software quality management process, LPH-GDO Model is implemented in java language with diverse size of software program code in the range of 10 KB to 100 KB. When performing the experimental process using 60 KB software program code size, proposed LPH-GDO Model attains95 %reliability whereas existing works NN-SQP technique [1] and Advanced Neural network with HCS [2] obtains 73% and 82 % respectively. Thus, reliability of software products using proposed LPH-GDO Model is very higher when compared to other conventional works. The performance result of reliability is portrayed in below Table 4.

Table 4 Tabulation for Reliability



Size of Software Program Code (KB)	Reliability (%)		
	Existing 1	Existing 2	Proposed Technique
10	69	74	88
20	67	78	89
30	68	77	91
40	71	80	90
50	69	79	93
60	73	82	95
70	70	85	92
80	72	87	94
90	74	86	93
100	78	89	96

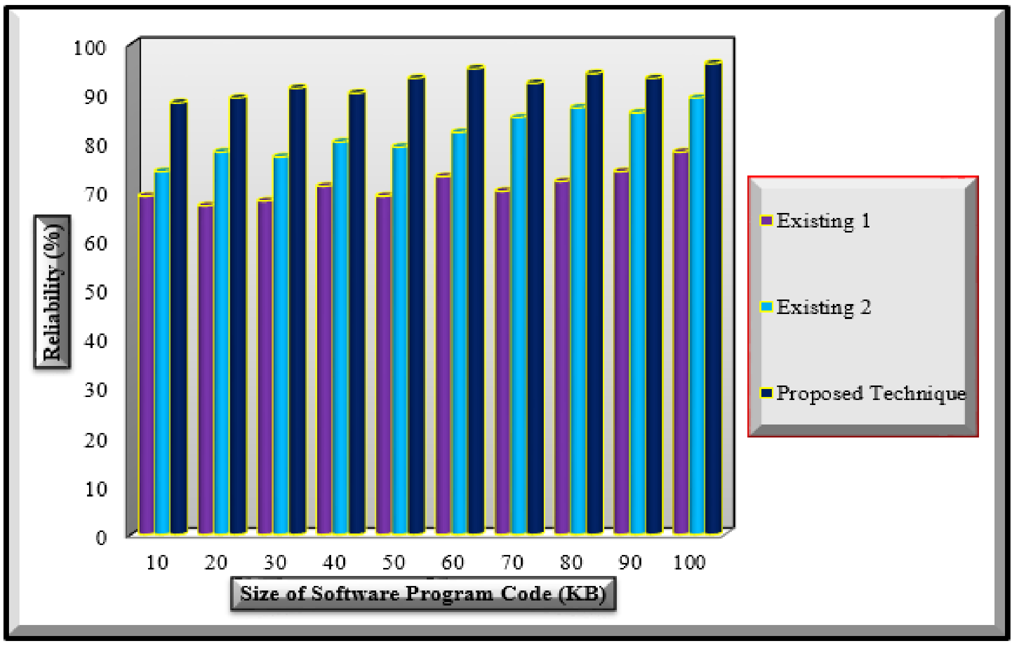


Figure 6 Comparative Result Analysis of Reliability versus Different Size of Software Program Code

Figure 6demonstrates the performance result analysis of reliability with respect tovaried size of software program code from schoolmate dataset using three methods namely, NN-SQP technique [1], Advanced Neural network with HCS [2] and proposed LPH-GDO Model. As depicted in above figure, proposed LPH-GDO Model provides better software reliability as compared to NN-SQP technique [1], Advanced Neural network with HCS [2] respectively. This is owing to application of Greedy Discritized Optimization algorithm where it measures testing cost and time ofallthe test cases in test suites. By using this, then Greedy Discritized Optimization algorithm builds the TR table. Next, Greedy Discritized Optimization algorithm identifies the optimal test cases based on objective function. At last, Greedy Discritized Optimization algorithm improves the reliability of software product viafinding the number of defects withselected optimal test cases. Hence, proposed LPH-GDO Model increasesreliability by 30 % and 13 % as NN-SQP technique [1], Advanced Neural network with HCS [2] respectively.

6. Conclusion

An efficientLPH-GDO Model is designed with the goal ofenhancing the scalability and reliability of software products with a minimal service provisioning time. The goal of LPH-GDO Model is obtained by application of Louvain Parallelization Heuristic algorithm and Greedy Discritized Optimization algorithm. By using the concepts of Louvain Parallelization Heuristic algorithm, LPH-GDO Modelperforms parallel implementation to obtain higher quality software program outputs in a less number of iterations when compared to state-of-the-art works. This supports for LPH-GDO Model to achieve higher scalability and to reduce the time required for providing user satisfied software program based on their requirements as compared to conventional works. Moreover, with the application of Greedy Discritized Optimization algorithm, LPH-GDO Model attain higher reliability of software products by finding the number of defects using optimal test cases as compared to existing works. From that, LPH-GDO Model gets higher software quality detection performance as compared to state-of-the-art works. The efficiency of LPH-GDO Model is tested with the parameters such as scalability, service provisioning time and software reliability. The experimental results show that LPH-GDO Model is provides better performance with an enhancement of software reliability and minimization of the service provisioning time as compared to the state-of-the-art works.

References

[1] Sadaf Sahar, Usman Qamar, Sadaf Ayaz, "Multilayer Neural Network and Fuzzy Logic Based Software Quality Prediction", World Academy of Science, Engineering and Technology International Journal of Computer and Systems Engineering, Volume 11, Issue 9, Pages 1024-1028, 2017

[2] Kavita Sheoran, Pradeep Tomar, Rajesh Mishra, "Software Quality Prediction Model with the Aid of Advanced Neural Network with HCS", Procedia Computer Science, Elsevier, Volume 92, Pages 418-424, 2016

[3] Chin-Yu Huang, Chih-Song Kuo, Shao-Pu Luan, "Evaluation and Application of Bounded Generalized Pareto Analysis to Fault Distributions in Open Source Software", IEEE Transactions on Reliability, Volume 63, Issue 1, Pages 309 – 319, March 2014

[4] Ayman Amin, Lars Grunske, Alan Colmana, "An approach to software reliability prediction based on time series modeling", Journal of Systems and Software, Elsevier, Volume 86, Issue 7, Pages 1923-1932, July 2013

[5] Ayanloye O. Shakiru, Mahmud Ahmad Yusuf, Koh Tieng Wei and Sukumar Letchmunan, "Software Quality: Predicting Reliability of a Software Using a Decision Tree", Information Technology Journal, Volume 13, Issue 18, Pages 2755-2759, 2014

[6] Arun Kumar Marandi, Danish AliKhan, "An Impact of Linear Regression Models for Improving the Software Quality with Estimated Cost", Procedia Computer Science, Elsevier, Volume 54, Pages 335-342, 2015

[7] Anu K P, BinuRajan, "A Novel Approach for Improving Software Quality Prediction", International Journal of Engineering and Advanced Technology (IJEAT), Volume 4 Issue 6, Pages 2249 – 8958, August 2015

[8] Trent A. Kroeger, Neil J. Davidson, Stephen C. Cook, "Understanding the characteristics of quality for software engineering processes: A Grounded Theory investigation", Information and Software Technology, Elsevier, Volume 56, Issue 2, Pages 252-271, February 2014

[9] Ivan Janicijevic, Maja Krsmanovic, Nedeljko Zivkovic, Sasa Lazarevic, "Software quality improvement: a model based on managing factors impacting software quality", Software Quality Journal, Springer, Volume 24, Issue 2, Pages 247–270, June 2016

[10] Alexandra Maria Simader, Bernhard Kluger, Nora Katharina Nicole Neumann, Christoph Bueschl, Marc Lemmens, Gerald Lirk, Rudolf Krska and Rainer Schuhmacher,"QCScreen: a software tool for data quality control in LC-HRMS based metabolomics", Springer, Volume 16, Volume 341, Pages 1-9, 2015

[11] Diwaker C, Tomar P, Poonia RC, Singh V, "Prediction of Software Reliability using Bio Inspired Soft Computing Techniques", Journal of Medical Systems, Elsevier, Volume 42, Issue 93, Pages 1-16, May 2018

[12] Miltiadis G.Siavvas, Kyriakos C.Chatzidimitriou, Andreas L.Symeonidis, "QATCH - An adaptive framework for software product quality assessment", Expert Systems with Applications, Elsevier, Volume 86, Pages 350-366, November 2017

[13] EkbalA. Rashid, Srikanta B. Patnaik and VandanaC. Bhattacherjee, "Machine Learning and Software Quality Prediction: As an Expert System", International Journal of Information Engineering and Electronic Business, Volume 2, Pages 9-27, 2014

[14] FernandoPinciroli, "Improving Software Applications Quality by Considering the Contribution Relationship among Quality Attributes", Procedia Computer Science, Elsevier, Volume 83, Pages 970-975, 2016

[15]D. Azar, J. Vybihal, "An ant colony optimization algorithm to improve software quality prediction models: Case of class stability", Information and Software Technology, Elsevier, Volume 53, Issue 4, Pages 388-393, April 2011

[16] wad Ali, Dayang N. A. Jawawi, Mohd Adham Isa,Muhammad Imran Babar, "Technique for Early Reliability Prediction of Software Components Using Behaviour Models", Plos One, Volume 11, Issue 12, Pages 1-24, 2016

[17] Fehmi Jaafar, Angela Lozano, Yann-Gaël Guéhéneuc, Kim Mens, "Analyzing software evolution and quality by extracting Asynchrony change patterns", Journal of Systems and Software, Volume 131, Pages 311-322, September 2017

[18] Deshan Cooray, Ehsan Kouroshfar, Sam Malek, Roshanak Roshandel, "Proactive Self-Adaptation for Improving the Reliability of Mission-Critical, Embedded, and Mobile Software", IEEE Transactions on Software Engineering, Volume 39, Issue 12, Pages 1714 – 1735, 2013

[19] Suejb, Sabri Pllana, Alécio Binotto, Joanna Kołodziej, Ivona Brandic, "Using meta-heuristics and machine learning for software optimization of parallel computing systems: a systematic literature review", Computing, Springer, Pages 1–44, 2018

[20] Harminder Pal Singh Dhami, "Improving Software Reliability, Productivity and Quality Using Software Metrics", International Journal of Computer Science and Technology, Volume 7, Issue 4, Pages 16-19, 2016