A Comparative Analysis of Conjugate Gradient Algorithms & PSO Based Neural Network Approaches for Reusability Evaluation of Procedure Based Software Systems

Parvinder S. Sandhu*[a], Shalini Chhabra [b]
[a] Computer Science & Engineering Department, Rayat-Bahra Institute of Engineering & Bio-Technology, Punjab, India.
[b] Computer Science & Engineering Department, Rayat Institute of Engineering & Information Technology, Rail Majra, Punjab, India.
*Author for correspondence; e-mail: sandhu@gmail.com

ABSTRACT

Software reusability is primary attribute of software quality. There are metrics for identifying the quality of reusable components but the function that makes use of these metrics to find reusability of software components is still not clear. These metrics if identified in the design phase or even in the coding phase can help us to reduce the rework by improving quality of reuse of the component and hence improve the productivity due to probabilistic increase in the reuse level. We have taken the dataset and framework of metrics, that make use of McCabe’s Cyclometric Complexity Measure for Complexity measurement, Regularity Metric, Halstead Software Science Indicator for Volume indication, Reuse Frequency metric and Coupling Metric values of the software component as input attributes and calculated reusability of the software component. In this work, Particle Swarm Optimization technique along with the four variants of Conjugate Gradient Algorithms is empirically explored to train a feed forward neural network for reusability dataset. The performance of the trained neural networks is tested to evaluate the reusability level of the procedure based software systems. The results are recorded in terms of Accuracy, Mean Absolute Error (MAE) and Root Mean Squared Error (RMSE).

Keywords: software reusability, particle swarm optimization, neural network.

1. INTRODUCTION

The demand for new software applications is currently increasing at the exponential rate, as is the cost to develop them. The numbers of qualified and experienced professionals required for this extra work are not increasing commensurably [1]. Software professionals have recognized reuse as a powerful means of potentially overcoming the above said software crisis [2, 3] and it promises significant improvements in software productivity and quality [4, 5].

There are two approaches for reuse of
code: develop the reusable code from scratch or identify and extract the reusable code from already developed code. The organization that has experience in developing software, but not yet used the software reuse concept, there exist extra cost to develop the reusable components from scratch to build and strengthen their reusable software reservoir [4]. The cost of developing the software from scratch can be saved by identifying and extracting the reusable components from already developed and existing software systems or legacy systems [6]. But the issue of how to identify reusable components from existing systems has remained relatively unexplored. In both the cases, whether we are developing software from scratch or reusing code from already developed projects, there is a need of evaluating the quality of the potentially reusable piece of software. The contribution of metrics to the overall objective of the software quality is understood and recognized [7-9]. But how these metrics collectively determine reusability of a software component is still at its early stage. A neural Network approach could serve as an economical, automatic tool to generate reusability ranking of software [10]. But, when one designs with Neural Networks alone, the network is a black box that needs to be defined, which is a highly compute-intensive process. One must develop a good sense, after extensive experimentation and practice, of the complexity of the network and the learning algorithm to be used.

In this paper, Particle Swarm Optimization trained Neural Network and Conjugate Gradient Algorithms based neural networks are empirically explored to evaluate the reusability of the function oriented software systems. This paper consists of 6 sections. The second section explains the related work regarding measuring the software quality specifically software reusability. The third section introduces the PSO technique used for the training of the neural network. In the fourth section the methodology of evaluating reusability is discussed. In the fifth section there implementation results are illustrated and in the final section conclusion is written.

2. RELATED WORK

One possible measure of a component’s reusability comes from its success; how many other applications modules access this common code? Other measures come from static code metrics. There are basically two approaches to evaluate software reusability: qualitative and empirical. The qualitative methods require substantial manual effort, as these methods generally rely on a subjective value attached to how well the software adheres to some guidelines or principles. On the other hand, empirical methods depend on the objective data that can be collected automatically and cheaply with the help of some tool [11].

Selby [12] identified a number of characteristics of those components, from existing systems, that are being reused at NASA laboratory and reported that the developers were successful in achieving a 32 percent reusability index. Selby’s recent experimental study has identified two categories of factors that characterize successful reuse-based software development of large-scale systems: module design factors and module implementation factors [13]. The module design factors that characterize module reuse without revision were: few calls to other system modules (i.e. low coupling), many calls to utility functions (i.e. high cohesion), few input-output parameters, few reads and writes, and many comments. The module implementation factors that characterize module reuse without revision were small in size (source lines) and with many
assignment statements (i.e. low Cyclometric Complexity). The modules reused without revision had the fewest faults, fewest faults per source line, and lowest fault correction effort. The modules reused with major revision had the highest fault correction effort.

Reformat et al have used decision tree based approach to the problems of identification of good or bad software based on Java and C++ objects. In the study fifteen metrics have been used and 55 to 72% accuracy has been reported [14]. Prieto-Diaz and Freeman encouraged white-box reuse and identified five program attributes for evaluating reusability [15]. The attributes used are:

- Program Size
- Program Structure
- Program Documentation
- Programming Language
- Reuse Experience

3. PARTICLE SWARM OPTIMIZATION

Particle swarm optimization (PSO) is a population based stochastic optimization technique developed by Eberhart and Kennedy in 1995, inspired by social behavior of bird flock or fish schooling [16]. The advantages of PSO are that PSO is easy to implement and there are few parameters to adjust. PSO has been successfully applied in many areas: function optimization, artificial neural network training, fuzzy system control, and other areas [16].

PSO is initialized with a group of random particles (solutions) and then searches for optima by updating generations. In every iteration, each particle is updated by following two “best” values. The first one is the best solution (fitness) it has achieved so far. (The fitness value is also stored.) This value is called pbest. Another “best” value that is tracked by the particle swarm optimizer is the best value, obtained so far by any particle in the population. This best value is a global best and called gbest. When a particle takes part of the population as its topological neighbors, the best value is a local best and is called lbest [16].

After finding the two best values, the particle updates its velocity and positions with following equation (3.1) and (3.2).

\[ v_i(t) = v_i(t) + c_1 \cdot r_1 \cdot (p_{best_i} - x_i(t)) + c_2 \cdot r_2 \cdot (g_{best} - x_i(t)) \]  \hspace{1cm} (3.1)

\[ x_i(t + 1) = x_i(t) + v_i(t) \]  \hspace{1cm} (3.2)

Where array \( v_i \) is the particle velocity and \( x_i(t) \) is the current particle (solution). The arrays pbest \( i \) and gbest \( i \) are defined as stated before. The function named as rand \( () \) is a random number between \((0, 1)\). The constants \( c_1, c_2 \) are learning factors. Usually \( c_1 = c_2 = 2 \).

As mentioned in [17], the pseudo code of the procedure is as follows:

For each particle
  Initialize particle
End

Do
  For each particle
    Calculate fitness value
    If the fitness value is better than the best fitness value (pbest) in history
      set current value as the new pbest
    End
  End
  Choose the particle with the best fitness value of all the particles as the gbest
For each particle
  Calculate particle velocity according equ. (3.1)
  Update particle position according equ. (3.2)
End
While maximum iterations or minimum error criteria is not attained Particles’ velocities on each dimension are clamped to a maximum velocity Vmax. If the sum of accelerations would cause the velocity on that dimension to exceed Vmax (which is a parameter specified by the user), then the velocity on that dimension is limited to Vmax.

4. METHODOLOGY

In the methodology following steps are used.

4.1 Metric Suit for Function Oriented Paradigm

We have taken the dataset and framework of metrics, discussed in our earlier papers [18] and [19]. A framework of real values of five input metrics is proposed for structural analysis of procedure or function-oriented. The output is the reusability value of the component. In the example dataset the reusability value is expressed in terms of levels of reusability values from 1 to 6. The value 1 means ‘Nil’ reusability level and value 6 shows the ‘Excellent’ reusability level. The code of software is parsed to calculate the metric values. The following suits of metrics are able to target those the essential attributes of function oriented features towards measuring the reusability of software modules, so it tried to analyze, refine and use following metrics to explore different structural dimensions of Function oriented components.

The proposed metrics for Function Oriented Paradigm are as follows.

4.1.1 Cyclometric Complexity Using McCabe’s Measure [18-20]

According to McCabe, the value of Cyclometric Complexity (CC) can be obtained using the following equation:

\[ CC = \text{Number of predicate nodes} + 1 \]  
(4.1)

Where predicate nodes are the nodes of the directed graph, made for the component, where the decisions are made.

4.1.2 Halstead Software Science Indicator [18, 19, 21]

According to this metric volume of the source code of the software component is expressed in the following equation:

\[ \text{Volume} = (N_1 + N_2) \log_2 (\eta_1 + \eta_2) \]  
(4.2)

Where, \( \eta_1 \) is the number of distinct operators that appear in the program, \( \eta_2 \) is number of distinct operands that appear in the program, \( N_1 \) is the total number of operator occurrences and \( N_2 \) is the total number of operand occurrences.

The high volume means that software component needs more maintenance cost, correctness cost and modification cost. On the other hand, less volume increases the extraction cost, identification cost from the repository and packaging cost of the component. So the volume of the reusable component should be in between the two extremes.

4.1.3 Regularity Metric [18, 19, 21]

The notion behind Regularity is to predict length based on some regularity assumptions. As actual length (N) is sum of \( N_1 \) and \( N_2 \). The estimated length is shown in the following equation:

\[ \text{Estimated Length} = \eta_1 \log_2 \eta_1 + \eta_2 \log_2 \eta_2 \]  
(4.3)

The closeness of the estimate is a measure of the Regularity of Component coding is calculated as:

\[ \text{Regularity} = 1 - \frac{(N - N')}{N} = \frac{N'}{N} \]  
(4.4)

Hence, there should be some minimum level of Regularity of the component to indicate the reusability of that component.

4.1.4 Reuse-Frequency Metric [18, 19, 21]

Reuse frequency is calculated by comparing number of static calls addressed
to a component with number of calls addressed to the component whose reusability is to be measured. Let N user defined components be X₁, X₂ … Xₙ in the system, where S₁, S₂ … Sₘ are the standard environment components e.g. printf in C language, then Reuse-Frequency is calculated as:

\[ \text{Reuse-Frequency} = \frac{n(C)}{1 + \sum_{i=1}^{m} n(S_i)} \]  

Equation (4.5) shows that the Reuse-Frequency is the measure of function usefulness of a component. Hence there should be some minimum value of Reuse-Frequency to make software component really reusable [22].

4.1.5 Coupling Metric [18, 19]

Functions/methods that are loosely bound tend to be easier to remove and use in other contexts than those that depend heavily on other functions or non-local data. Different types of coupling effects reusability to different extent.

Weight of coupling increases from category “a” to “d”, means

Data Coupling is lightest weight coupling, whereas Content Coupling is the heaviest one. Let

\[ a_i \]  
\[ b_i \]  
\[ c_i \]  
\[ d_i \]  

be the number of functions called and Data Coupled with function “i”

be the number of functions called and Stamp Coupled with function “i”

be the number of functions called by function “i” and Control Coupled with function “i”

be the number of functions Common Coupled with function “i”

\[ f(a, b, c) = \frac{1}{1 + e^{-a(b+c)+w(a+b)+w(b+c)}} \]  

Where \( a = 10, c = 0.5 \) and \( w_i \) for \( i = 1, 2, 3, 4 \) is the weights of the respective the coupling types.

As coupling increases, there is decrease in understandability and maintainability, so there should be some maximum value of the coupling.

4.2 Design and Evaluate Neural Network System

The following five Neural Network algorithms are experimented:

- Fletcher–Reeves Update Conjugate Gradient (FRUCG) algorithm
- Polak–Ribiere Update Conjugate Gradient (PRUCG) algorithm
- Powell–Beale Restarts Conjugate Gradient (PBRCG) algorithm
- Scaled Conjugate Gradient (SCG) algorithm
- Neural Network Trained with particle swarm optimization Algorithm (NPSO)

Once the network weights and biases are initialized, the network is ready for training. The network can be trained for function approximation (nonlinear regression), pattern association, or pattern classification. The training process requires a set of examples of proper network behavior — network inputs \( p \) and target outputs \( t \). During training the weights and biases of the network are iteratively adjusted to minimize the network performance function.

The basic backpropagation algorithm adjusts the weights in the steepest descent direction (negative of the gradient), the direction in which the performance function is decreasing most rapidly. It turns out that, although the function decreases most rapidly along the negative of the gradient, this does not necessarily produce the fastest convergence. In the conjugate gradient algorithms a search is performed along conjugate directions, which produces generally faster convergence than steepest descent.
directions. In most of the training algorithms, a learning rate is used to determine the length of the weight update (step size) and the step size is adjusted at each iteration. Here, a search is made along the conjugate gradient direction to determine the step size that minimizes the performance function along that line. There are four variations of conjugate gradient algorithms:

a) Fletcher-Reeves Update [22-24]

This is the ratio of the norm squared of the current gradient to the norm squared of the previous gradient. The algorithm can train any network as long as its weight, net input, and transfer functions have derivative functions. Backpropagation is used to calculate derivatives of performance with respect to the weight and bias variables X. Each variable is adjusted according to the following:

\[ X = X + a \cdot dX \] (4.7)

Where, \( dX \) is the search direction. The parameter ‘a’ is selected to minimize the performance along the search direction. The line search function is used to locate the minimum point. The first search direction is the negative of the gradient of performance. In succeeding iterations the search direction is computed from the new gradient and the previous search direction according to the formula:

\[ dX = gX + dX_{old} \cdot Z \] (4.8)

Where, \( gX \) is the gradient. The parameter ‘Z’ can be computed in several different ways. For the Fletcher-Reeves variation of conjugate gradient, it is computed according to the following:

\[ Z = \frac{\left([gX - gX_{old}] \cdot gX\right)}{\text{norm}_sqr} \] (4.9)

Where, \text{norm}_sqr is the norm square of the previous gradient and \text{normnew}_sqr is the norm square of the current gradient.

b) Polak-Ribiôte Update [22-24]

Another version of the conjugate gradient algorithm was proposed by Polak and Ribiôte. The algorithm can train any network as long as its weight, net input, and transfer functions have derivative functions. Here also, backpropagation is used to calculate derivatives of performance with respect to the weight and bias variables X. Each variable is adjusted according to the equation (4.7). The parameter ‘a’ is selected to minimize the performance along the search direction. The line search function is used to locate the minimum point. The first search direction is the negative of the gradient of performance. In succeeding iterations the search direction is computed from the new gradient and the previous search direction according to the equation (4.8). Where, \( gX \) is the gradient. The parameter ‘Z’ can be computed in several different ways. For the Polak-Ribiôte variation of conjugate gradient, it is computed according to the following equation:

\[ Z = \frac{\left([gX - gX_{old}] \cdot gX\right)}{\text{norm}_sqr} \] (4.10)

Where, \text{norm}_sqr is the norm square of the previous gradient, and \( gX_{old} \) is the gradient on the previous iteration. The storage requirements for Polak-Ribiôte (four vectors) are slightly larger than for Fletcher-Reeves (three vectors).

c) Powell-Beale Restarts

For all conjugate gradient algorithms, the search direction is periodically reset to the negative of the gradient. The standard reset point occurs when the number of iterations is equal to the number of network parameters (weights and biases), but there are other reset methods that can improve the efficiency of
training. One such reset method was proposed by Powell [25], based on an earlier version proposed by Beale [26]. This technique restarts if there is very little orthogonality left between the current gradient and the previous gradient. This is tested with the following inequality:

\[ g_1^T (g_1 - g_0) \geq 0.2 ||g_1||^2 \]  \hspace{1cm} (4.11)

If this condition is satisfied, the search direction is reset to the negative of the gradient.

This algorithm can train any network as long as its weight, net input, and transfer functions have derivative functions. Backpropagation is used to calculate derivatives of performance with respect to the weight and bias variables \(X\). Each variable is adjusted according to equation (4.7). The parameter “\(a\)” is selected to minimize the performance along the search direction. The line search function is used to locate the minimum point. The first search direction is the negative of the gradient of performance. In succeeding iterations the search direction is computed from the new gradient and the previous search direction according to equation (4.8). Where \(gX\) is the gradient. The parameter “\(Z\)” can be computed in several different ways. So, The Powell-Beale variation of conjugate gradient is distinguished by two features. First, the algorithm uses a test to determine when to reset the search direction to the negative of the gradient. Second, the search direction is computed from the negative gradient, the previous search direction, and the last search direction before the previous reset.

The storage requirements for the Powell-Beale algorithm (six vectors) are slightly larger than for Polak-Ribiére (four vectors).

d) Scaled Conjugate Gradient

Each of the conjugate gradient algorithms discussed so far requires a line search at each iteration. This line search is computationally expensive, because it requires that the network response to all training inputs be computed several times for each search. The scaled conjugate gradient algorithm (SCG), developed by Moller [27], was designed to avoid the time-consuming line search. This algorithm combines the model-trust region approach with the conjugate gradient approach [27]. The storage requirements for the scaled conjugate gradient algorithm are about the same as those of Fletcher-Reeves.

The algorithm train any network as long as its weight, net input, and transfer functions have derivative functions. Backpropagation is used to calculate derivatives of performance with respect to the weight and bias variables \(X\). The scaled conjugate gradient algorithm is based on conjugate directions, but this algorithm does not perform a line search at each iteration.

The following are the phases for the different Neural Network based modeling system:

4.2.1 Phase I

In this phase Designing of Neural Network and Training is performed. The following sub steps are used:

- Load the data
- Divide data into Training, Validation and Test data
- Calculate the minimum and maximum values in the attribute of input and setting the various parameters of feedforward back-propagation network by like:
  - Size of the Neural Network
  - Type of Transfer function of each layer to be used
  - Type of learning function
  - Generate the Neural Network
  - Perform the training of the Neural
Network using the training dataset. Training is accomplished by sending a given set of inputs through the network and comparing the results with a set of target outputs.

- If there is a difference between the actual and target outputs, the weights are adjusted to produce a set of outputs closer to the target values.
- Network weights are determined by adding an error correction value to the old weight and the biasness value is also adjusted. Training stops when any of these conditions occurs:
  - The maximum number of epochs (repetitions) is reached.
  - The maximum amount of time is exceeded.
  - Performance is minimized to the goal.
  - The performance gradient falls below minimum value set.
  - Validation performance has increased more than maximum fail times since the last time it decreased (when using validation).

### 4.2.2 Phase II

This phase is a Testing phase. In this step the trained Neural Network is evaluated against the testing data on the different criteria as described in the next step.

### 4.3 Comparison Criteria

The comparisons are made on the basis of value of MAE, RMSE and Accuracy values of the trained neural network model. Different iterations are taken from the data set then testing is performed and the average value is taken as show in the table1 in results. The details of the MAE and RMSE are given below:

#### 4.3.1 Mean Absolute Error (MAE)

Mean absolute error, MAE is the average of the difference between predicted and actual value in all test cases; it is the average prediction error [28]. The formula for calculating MAE is given in equation shown below:

$$MAE = \frac{|a_1 - c_1| + |a_2 - c_2| + \ldots + |a_n - c_n|}{n} \quad (4.7)$$

Assuming that the actual output is a, expected output is c.

#### 4.3.2 Root Mean-Squared Error (RMSE)

RMSE is frequently used measure of differences between values predicted by a model or estimator and the values actually observed from the thing being modeled or estimated [29]. It is just the square root of the mean square error as shown in equation given below:

$$RMSE = \sqrt{\frac{(a_1 - c_1)^2 + (a_2 - c_2)^2 + \ldots + (a_n - c_n)^2}{n}} \quad (4.8)$$

### 5 Implementation and Results

In this paper, the implementation of the algorithm is done in Matlab7.4 environment and the PSO trained Neural Network particle swarm optimization toolbox for Matlab [30] is used. The dataset is collected and divided into training and testing exemplars.

Size of the feed-forward backpropagation Neural Network is set as [5 5 1] means there are 5 neurons in the Input layer, 5 neurons in the hidden layer and one neuron in the output layer of the network. The Linear transfer function is the type of Transfer function used for the last layer and Hyperbolic tangent sigmoid transfer function is used for the rest of layer of the designed neural network. Gradient descent with momentum weight and bias learning function is used as Back propagation weight/bias learning
function.

TRAINPSO type of Back-propagation network training function is used as TRAINPSO is a network training function that updates weight and bias values according to particle swarm optimization. The following are the additional parameters values used:

- \text{maximum iterations: 200}
- \text{population size: 25}
- \text{acceleration constants (for type = 0): [2,2]}
- \text{inertia weights (for type = 0): [0.9,0.4]}
- \text{minimum error gradient: 1e-9;}
- \text{Iterations at error grad value before exit: floor(0.2*trainParam.maxit)}
- \text{error goal: 0;}
- \text{Type of PSO: Trelea}

After the training, the Architecture of Feed Forward Neural Network is shown in Figure 1 where the Bright Green line shows more positive weight, bright red line shows more negative weight and dashed white line shows zero weight i.e. no connection between the neurons. In the testing phase \text{MAE}, \text{RMSE} and \text{Accuracy} values of the system are 0.20816, 0.24007, and 74% respectively as shown in Table 1.

\textbf{Figure 1.} Architecture of feed forward neural network.
During the testing of the developed system 0.20816, 0.24007 and 74 are calculated as MAE, RMSE and Accuracy values for the testing dataset as shown in Table 1.

**Table 1.** Iteration wise Results of the NPSO system on the basis of the proposed criteria.

<table>
<thead>
<tr>
<th>Iterations no.</th>
<th>Accuracy</th>
<th>MAE</th>
<th>RMSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>70</td>
<td>0.2121</td>
<td>0.2576</td>
</tr>
<tr>
<td>2</td>
<td>80</td>
<td>0.1638</td>
<td>0.1965</td>
</tr>
<tr>
<td>3</td>
<td>100</td>
<td>0.1190</td>
<td>0.1429</td>
</tr>
<tr>
<td>4</td>
<td>70</td>
<td>0.2263</td>
<td>0.2574</td>
</tr>
<tr>
<td>5</td>
<td>70</td>
<td>0.2267</td>
<td>0.2578</td>
</tr>
<tr>
<td>6</td>
<td>70</td>
<td>0.2287</td>
<td>0.2601</td>
</tr>
<tr>
<td>7</td>
<td>70</td>
<td>0.2253</td>
<td>0.2564</td>
</tr>
<tr>
<td>8</td>
<td>70</td>
<td>0.2250</td>
<td>0.2562</td>
</tr>
<tr>
<td>9</td>
<td>70</td>
<td>0.2255</td>
<td>0.2566</td>
</tr>
<tr>
<td>10</td>
<td>70</td>
<td>0.2292</td>
<td>0.2592</td>
</tr>
<tr>
<td>Average</td>
<td>74</td>
<td>0.20816</td>
<td>0.24007</td>
</tr>
</tbody>
</table>

The Table 2 shows the Iteration-wise, 5 iterations, testing results of Fletcher–Reeves Update Conjugate Gradient (FRUCG) algorithm, Polak -Ribiere update Conjugate Gradient (PRUCG) algorithm, Powell -Beale Restarts Conjugate Gradient (PBRCG) algorithm and Scaled Conjugate Gradient (SCG) algorithm in terms of Accuracy, MAE and RMSE values.

**Table 2.** Iteration-wise testing results of conjugate gradient algorithms conjugate gradient algorithms.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Iterations no.</th>
<th>Accuracy</th>
<th>MAE</th>
<th>RMSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>FRUCG</td>
<td>1</td>
<td>80</td>
<td>0.0458</td>
<td>0.0605</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>80</td>
<td>0.0414</td>
<td>0.0523</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>70</td>
<td>0.0439</td>
<td>0.0508</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>80</td>
<td>0.0305</td>
<td>0.0444</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>80</td>
<td>0.0414</td>
<td>0.0481</td>
</tr>
<tr>
<td>PRUCG</td>
<td>1</td>
<td>70</td>
<td>0.0418</td>
<td>0.0490</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>60</td>
<td>0.0514</td>
<td>0.0612</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>60</td>
<td>0.0577</td>
<td>0.0700</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>80</td>
<td>0.0399</td>
<td>0.0457</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>70</td>
<td>0.0488</td>
<td>0.0542</td>
</tr>
<tr>
<td>PBRCG</td>
<td>1</td>
<td>70</td>
<td>0.0435</td>
<td>0.0496</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>70</td>
<td>0.0520</td>
<td>0.0608</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>70</td>
<td>0.0500</td>
<td>0.0593</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>70</td>
<td>0.0490</td>
<td>0.0574</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>80</td>
<td>0.0367</td>
<td>0.0499</td>
</tr>
<tr>
<td>SCG</td>
<td>1</td>
<td>70</td>
<td>0.0407</td>
<td>0.0496</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>90</td>
<td>0.0409</td>
<td>0.0475</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>80</td>
<td>0.0383</td>
<td>0.0513</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>90</td>
<td>0.0391</td>
<td>0.0483</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>90</td>
<td>0.0402</td>
<td>0.0451</td>
</tr>
</tbody>
</table>

The training Performance of Fletcher–Reeves Update Conjugate Gradient (FRUCG) algorithm, Polak -Ribiere update Conjugate Gradient (PRUCG) algorithm, Powell -Beale Restarts Conjugate Gradient (PBRCG) algorithm and Scaled Conjugate Gradient (SCG) algorithm is shown with help of graph.
Figure 3. Training performance of Fletcher-Reeves update conjugate gradient algorithm.

Figure 4. Training performance of Polak-Ribiere update conjugate gradient (PRUCG) algorithm.

Figure 5. Training performance of Powell-Beale restarts conjugate gradient (PBRCG) algorithm.

Figure 6. Training performance of scaled conjugate gradient (SCG) algorithm.

The Table 3 shows the average accuracy, MAE and RMSE values of 5 iterations testing results of Fletcher-Reeves Update Conjugate Gradient (FRUCG) algorithm, Polak-Ribiere update Conjugate Gradient (PRUCG) algorithm, Powell-Beale Restarts Conjugate Gradient (PBRCG) algorithm and Scaled Conjugate Gradient (SCG) algorithm.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Mean Accuracy</th>
<th>Mean MAE</th>
<th>Mean RMSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>FRUCG</td>
<td>78</td>
<td>0.0406</td>
<td>0.05122</td>
</tr>
<tr>
<td>PRUCG</td>
<td>68</td>
<td>0.04792</td>
<td>0.05602</td>
</tr>
<tr>
<td>PBRCG</td>
<td>72</td>
<td>0.04624</td>
<td>0.0554</td>
</tr>
<tr>
<td>SCG</td>
<td>84</td>
<td>0.03984</td>
<td>0.0484</td>
</tr>
<tr>
<td>NN-PSO</td>
<td>74</td>
<td>0.20816</td>
<td>0.24007</td>
</tr>
</tbody>
</table>
6. CONCLUSION

In this paper, it is Feed Forward Neural Network trained with Particle Swarm Optimization Algorithm and four types of Conjugate Gradient algorithms are used to evaluate the reusability of procedure oriented software systems. The performance of Feed Forward Neural Network trained with Scaled Conjugate Gradient (SCG) algorithm comes out to be the best with $MAE$, $RMSE$ and $Accuracy$ values of 0.03984, 0.0484 and 84% respectively and it has outperformed the three other types of Conjugate Gradient algorithms as well as PSO trained Neural Network. The results obtained using proposed system is better than the results mentioned in literature [18] and [19]. The second best performance is of Fletcher–Reeves Update Conjugate Gradient (FRUCG) algorithm with $MAE$, $RMSE$ and $Accuracy$ values of 0.0406, 0.05122 and 78% respectively. Hence, the proposed system can be used for the evaluation of reusability of the procedure based systems.

The future work can be extended in following directions:

- This work can be extended to other programming languages.
- More algorithms can be evaluated and then we can find the best algorithm.
- Other dimensions of quality of software can be considered for mapping the relation of attributes.

REFERENCES


