Extended COCOMO: robust and interpretable neuro-fuzzy modelling

Brajesh Kumar Singh

Department of Computer Science and Engineering,
Raja Balwant Singh Engineering Technical Campus,
Agra, India
Email: brajesh1678@gmail.com

Shailesh Tiwari*

Department of Computer Science and Engineering,
ABES Engineering College,
Ghaziabad, India
Email: shail.tiwari@yahoo.com
*Corresponding author

Krishn Kumar Mishra

Department of Computer Science and Engineering,
MNNIT Allahabad,
Prayag, India
Email: kkm@mnnit.ac.in

Akash Punhani

Department of Computer Science and Engineering,
ASET,
Amity University,
Noida, India
Email: akashpunhani@gmail.com

Abstract: Prediction of software development efforts is one of the crucial activities in software project management. Still, search for the perfect model for software cost estimation has become most difficult task of the organisations dealing in software development. This paper presents the extended version of COCOMO, which is done with the help of two very popular methods, i.e., artificial neural networks (ANN) and fuzzy logic. Firstly, the expert judgement about model is used for validation, and overpowers the common software engineering 'black box' problem that arises widely in ANN-based solutions. Moreover, we choose the best combination of one of the three membership functions for continuous-rating values, which reduce the variance while estimating the cost of similar projects. The validation, using 93 NASA projects dataset, shows that the model significantly improves the estimation accuracy in terms of mean magnitude of relative error (MMRE) by 10.104% relative to other known estimation models.

Keywords: fuzzy logic; neural network; constructive cost model; COCOMO; neuro-fuzzy software effort estimation; NASA projects dataset; mean magnitude of relative error; MMRE.

Reference to this paper should be made as follows: Singh, B.K., Tiwari, S., Mishra, K.K. and Punhani, A. (2021) 'Extended COCOMO: robust and interpretable neuro-fuzzy modelling', *Int. J. Computational Vision and Robotics*, Vol. 11, No. 1, pp.41–65.

Biographical notes: Brajesh Kumar Singh completed his Doctorate degree in Computer Science and Engineering from Motilal Nehru National Institute of Technology, Allahabad (U.P.) in 2014. Presently, he is working as a Professor and the Head of The Department in CSE at R.B.S. Engineering Technical Campus, Agra, UP., India. He is supervising two PhD candidates. He has more than 50 publications to his credit in national and international journals and proceedings with large number of citations. He has delivered several key note addresses in national and international conferences. He has organised more than 40 international and national conferences in India and abroad.

Shailesh Tiwari is currently working as a Professor in Computer Science and Engineering Department, ABES Engineering College, Ghaziabad, India. He is an alumnus of Motilal Nehru National Institute of Technology Allahabad, India. His primary areas of research are software testing, implementation of optimisation algorithms and machine learning techniques in engineering problems. He has published more than 50 publications in international journals and in proceedings of international conferences of repute. He has edited special issues of Scopus, SCI and E-SCI-indexed journals. He has organised several international conferences under the banner of IEEE and Springer. He is a senior member of IEEE, member of IEEE Computer Society, and Fellow of Institution of Engineers (FIE).

Krishn Kumar Mishra is currently working as Assistant Professor, Department of Computer Science and Engineering, Motilal Nehru National Institute of Technology Allahabad, India. He has also been worked as a Visiting Faculty in Department of Mathematics and Computer Science, University of Missouri, St. Louis, USA. His primary area of research includes evolutionary algorithms, optimisation techniques and design, and analysis of algorithms. He has also published more than 50 publications in international journals and in proceedings of internal conferences of repute. He is serving as a program committee member of several conferences and also edited Scopus and SCI-indexed journals. He is also a member of reviewer board of *Applied Intelligence Journal*, Springer.

Akash Punhani has completed his Doctorate degree in Computer Science and Engineering from Jaypee University of Information technology, Waknaghat Solan Himachal Pradesh. Presently, he is working as an Assistant Professor III in Department in CSE at Amity University, Noida, UP, India. He is supervising two MTech candidates. He has published articles in various reputed journals listed in Scopus and SCI. His broad area of research includes optimisation problems, machine learning algorithms and interconnection networks.

1 Introduction

The fineness of the software developed and the yield of the software development process depend on the perfect estimation of production cost and causes that affect the development environment. Perfect estimation can be achieved by well-understood and closely linked software development process with software estimation process (Yu, 1990).

Many software estimation models (Boehm et al., 1981; Matson et al., 1994; Pillai and Nair, 1997; Shepperd and Kadoda, 2001) have been developed over the last few decades. Still, finding the right model for software cost estimation has become most difficult task of the organisations dealing in software development. Heemstra (1992) highlighted the importance of estimation models and there reliability. Constructive cost model (COCOMO) is among one of popular approach used in recent years. In this paper, an extended version of COCOMO has been used. This extension is done with the help of two very popular tactics, i.e., ANN and fuzzy logic, which ultimately provide the foundation for estimation models (Dote and Ovaska, 2001; Mitra and Hayashi, 2000) by avoiding the problem of large variance in cost estimations of similar projects (Garratt and Hodgkinson, 1999).

In the last three decades most estimation models are either regression models also known as parametric models or soft computing-based models. In the 1980s, the parametric models (Albrecht, 1979; Albrecht and Gaffney, 1983; Kemerer, 1987; Putnam, 1978) were widely used for comparing datasets of numerous dimensions in different situations with poor predictions (Kemerer, 1987; Kitchenham and Taylor, 1985). Kemerer (1987) analysed 15 business applications using four models: COCOMO (Boehm et al., 1981), estimacs (Putnam, 1978), function points (Albrecht and Gaffney, 1983), and SLIM (Putnam, 1978). Error estimation was reported in range of 72% to 85% of MMRE. Tests were made on three models: SLIM, COCOMO, and Jensen's (1983) model and reported MMRE in the range of 70% to 90%.

During the 1990s, non-parametric modelling techniques such as machine learning algorithms and analogy were also included in comparative studies of software estimation. Shepperd and Schofield (1997) studied analogy-based modelling technique and found it better in comparison to stepwise regression analysis in cases in terms of error. Mukhopadhyay and Kekre (1992) worked on same project dataset which was used by Kemerer and found that case-based reasoning (CBR) analogy model is better than other analogy models, performed better than COCOMO. Finnie et al. (1997) study various models using regression techniques using functional point (FP), ANN and CBR on data collected from 17 organisations with total of 299 projects. The study reveals that CBR has performed better than all other techniques under consideration. Srinivasan and Fisher (1995) took account of comparison in: regression trees, FP, ANN, COCOMO, and SLIM. The performance of regression trees was better than COCOMO and SLIM and found that function point-based artificial neural networks (ANN) prediction models performed better than regression trees. Briand et al. (1992) compared the COCOMO, stepwise regression, and machine learning-based optimised set reduction (OSR) non-parametric technique based on COCOMO and the Kemerer datasets. OSR performed significantly better than other two. Jorgensen (1995) applied various variable regression equations, numerous variants of ANN, and blends of OSR with regression techniques on 100 projects for accurate model development. He found that it is good to use two models

of multiple regressions but the fusion of OSR and regression is best. In general, Jorgensen (1995) recommended that more refined estimation models such as OSR along with expert judgment should be used to ascertain the funds and resources in those models. Although parametric techniques are considered in almost all of the research investigations based on comparison between various cost estimation methods, but their results are not always same and showing variations as per used techniques. These variations in the results various methods are due to the difference in the experimental designs of the models, even the same dataset is used in all those methods. Moreover, in many studies, small datasets from different software development environments have been used. Above difficulties made tough to conclude which models is efficiency. Our research work includes contributions of assessing and comparing most of the commonly used cost modelling techniques with company specific and multi-organisational data in software engineering and project management, which allows the relative utility of multi-organisational databases to be evaluated. In Boehm (2017) highlighted the 4P's (product, process, property, and personnel options) that makes this challenge more tough for the prediction of exact effort estimation. Till now, various advance prediction models are explored to predict the actual cost of the effort required. Ghatasheh et al. (2019) suggested the use of firefly algorithm for the software cost estimation. Mustapha et al. (2019) suggested the used of advance decision making technique random forest for the prediction. Venkataiah et al. (2019) suggested the use of spiking neural network to help in the estimation. Still there is a need for the perfect model to be developed for the prediction of the effort and cost estimation. A similar hybrid model based on the COCOMO and neuro-fuzzy has been proposed to get the cost estimation. The same have been evaluated on the popular metrics that are mean magnitude of relative error (MMRE) and *PRED*(*k*).

The following paper has been divided into six sections. The current section introduces the software cost estimation and its importance. Section 2 of the paper describes the problem statement and research methodology used to solve the problem. In Section 3, the proposed approach is being discussed. Section 4 describes the matrices used for the evaluation. The results obtained are elaborated in Section 5 and finally Section 6 concludes the article.

2 Problem description and research methodology

The several cost estimation techniques which are being used, may be clustered in following two most important categories:

1 statistical models

2 soft computing techniques-based models

2.1 Statistical models

These models are designed on the basis of statistical study of the past software projects like COCOMO (Boehm et al., 1981), Putnam's (1978) SLIM and Albrecht's function point methods (Kan, 2002) uses previous projects and are using simple linear regression. COCOMO and SLIM, both take project size in lines of code as the major input to their

models which is least known at the early stage in the project. These models have three major limitations:

1 Attributes and their relationships used to predict software development effort are time dependent and/or may differ for software development environments (MacDonell and Gray, 1998).

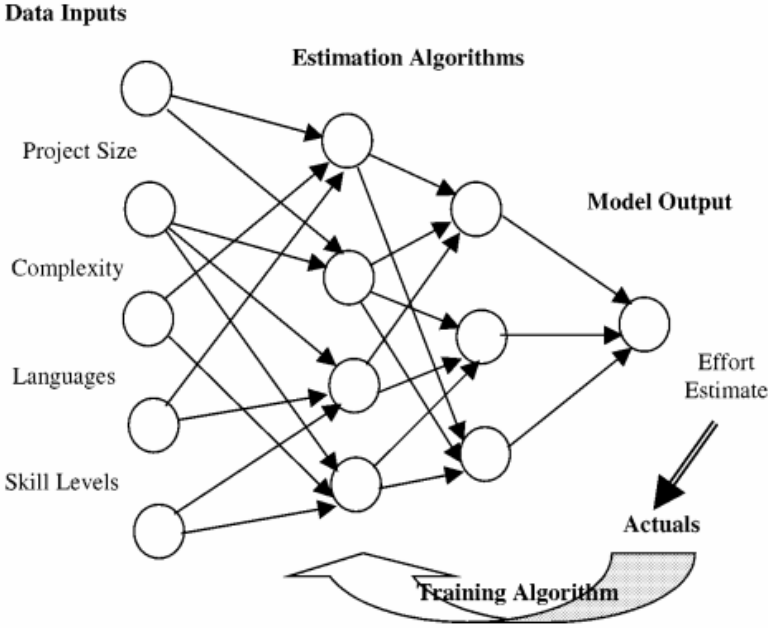
2 Accurate software size estimation in terms of source lines of code (SLOC), number of user screens, interfaces, complexity, etc., are the parameters desirable in existing statistical models at very early in development process when uncertainty surrounds the most.

3 Lack of categorical data handling capabilities and most importantly lacking logical reasoning capabilities and their inability to draw some valid conclusions or to take appropriate decisions based on recently available data (MacDonell and Gray, 1998).

2.2 Soft computing techniques-based models

To avoid the weaknesses of statistical models, many studies contributed to explore soft computing-based models (Du et al., 2015; Jodpimai et al., 2010; Malathi and Sridhar, 2011; Sandhu et al., 2008; Saxena and Singh, 2012). These models have the capabilities to learn from historical data to adequately model the complex relationship between multiplicative factors (cost drivers).

Figure 1 A neural network estimation model



According to Gray and MacDonell (1997), neural network is most commonly used software estimation model building technique in alternative to mean least squares regression technique. Neural networks works on the concept of 'training' in which experience data is used for learning. In the context of software estimation, Gray described one of the various types of neural network, a 'back-propagation trained feed-forward' network, which is the most widely used form of a neural network (Gray and MacDonell, 1997) as shown in Figure 1. From Figure 1, it can be observed that five parameters are provided to the input layer of the neural network and one output is generated.

In the development of such a neural network-based model, first step is to develop an appropriate structured form consisting of neurons, or network connections using nodes by based upon the various layers, i.e., input layer, hidden layers and output layer, number of neurons within each layer, and also the type of arrangement of linking these neurons. Once the network has been established, next step is to train the model by producing historical set of project data inputs with available actual values of project cost. A weighted sum of corresponding inputs is computed by each neuron in the network and an output, above certain threshold is provided. The network model then iterates according to the specified training algorithm. The delta value specification is important factor to avoid over training of known historical data. Some of the very early works including those of Boehm et al. (2000) and Kumar et al. (1994) indicate that the neural networks are highly applicable to the software cost estimation. Many researchers used neural networks in cost estimation, which are reported in Idri et al. (2007), Li (2010), Sheta et al. (2008), Venkatachalam (1993) and Vijayakumar (1997).

Since its foundation by Zadeh in 1965, fuzzy logic has been a valuable concept in recent time to solve highly complex problems where the mathematical modelling of these problems is too difficult or impossible. It is also helpful in reduction of the difficulty of prevailing results and escalation in the availability of control systems-based theories. Description of human reasoning-based logic is sometime inadequate for two truth-values, i.e., true ('1') and false ('0'). Such type of human reasoning can be solved by fuzzy logic, which maps an input space to output space and uses the whole interval between 0 and 1.

2.2.1 Fuzzy set and membership functions

Fuzzy set is a set, in which at least one element have fractional degrees of membership. It is defined on a universe of discourse (UOD) χ, and maps the element of χ into set members which denote the membership of the UOD in the set. Membership function is a mathematical expression to specify rate that a particular element belong to particular a fuzzy set (Zadeh, 1994).

A fuzzy set is normalised if its largest membership value equals 1.Classical sets admit memberships of either 0 or 1. We therefore have membership function $\mu_A(x)$ of x in a classical set A, as a subset of UOD χ, as described by equation (1):

$$\mu_A(x) = \begin{Bmatrix} 1 & \text{if} & x \in A \\ 0 & \text{if} & x \notin A \end{Bmatrix} \tag{1}$$

This is where fuzzy set comes into action. Fuzzy sets admit continuum memberships between 0 and 1 as represented in the equation (2):

$$\mu_A(x) : \chi \to [0, 1] \tag{2}$$

In practice, membership functions usually take on one of the three different standard equations and shapes, i.e., triangular, trapezoidal, and Gaussian.

1 *Triangular:* Figure 2 describes the graphical representation of the membership function and the equation based on the figure for the membership is described by equation (3). Figure 2 shows that membership function is convex membership function as it increase initially and then decreases keeping the monotonicity.

$$\mu(x) = \begin{cases} \dfrac{x-a}{b-a} & \text{if} \quad x \in (a, b) \\ \dfrac{c-x}{c-b} & \text{if} \quad x \in (b, c) \\ 0 & \quad otherwise \end{cases} \tag{3}$$

2 *Trapezoidal:* Figure 3 describes the graphical representation of the membership function and the equation based on the figure for the membership is described by equation (4). Figure 3, it can be observed that the core is the range form *b* to *c*.

$$\mu(x) = \begin{cases} \dfrac{x-a}{b-a} & \text{if} \quad x \in (a, b) \\ 1 & \text{if} \quad x \in (b, c) \\ \dfrac{c-x}{c-b} & \text{if} \quad x \in (c, d) \\ 0 & \quad otherwise \end{cases} \tag{4}$$

3 *Gaussian:* Figure 4 describes the graphical representation of the membership function and the equation based on the figure for the membership is described by equation (5). The parameter *a* and *b* are used to describe the shape of the Gaussian functions.

$$\mu(x) = \exp\left(-\frac{(x-a)^2}{2b^2}\right) \tag{5}$$

Figure 2 Describes the triangular membership function (see online version for colours)

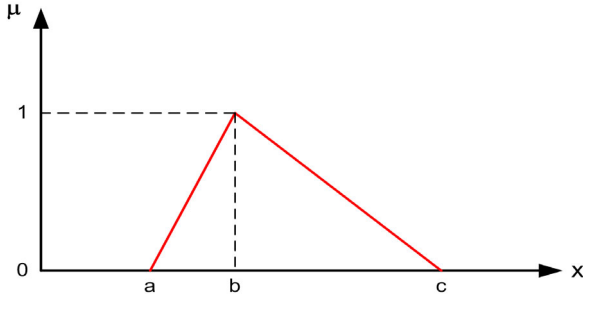


Figure 3 Describes the trapezoidal membership function (see online version for colours)

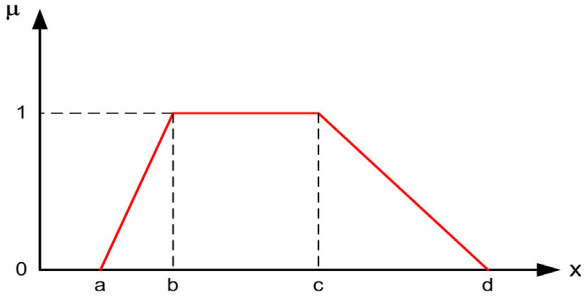
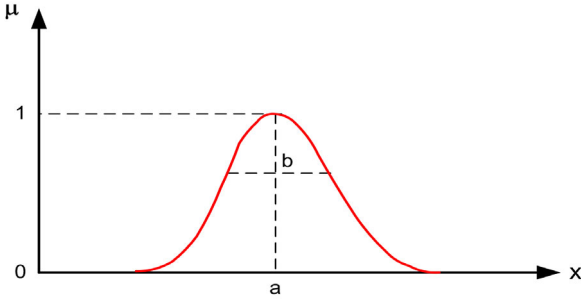


Figure 4 Describes the Gaussian membership function (see online version for colours)



3 Proposed approach

3.1 Architecture of the proposed approach

The proposed integrated neuro-fuzzy approach for software effort estimation is represented in Figure 5. The integrated neuro-fuzzy approach is consisting of two major components:

- *Fifteen sub-models (NF_i):* there are 15 independent sub models. For every sub-model, input value is used for cost driver ratings, and output is for revised effort multiplier value (REM_i). The REM_i is used as the input to the integrated approach of function point and COCOMO.
- *Integrated approach:* input belongs to the software size, values of corresponding projects modes and the output is the value of NF_i in terms of revised effort multipliers. Resultant value represents software effort estimation.

3.2 Sub-model NF_i

The adaptive neuro-fuzzy inference system (ANFIS) (Jang, 1993; Shi et al., 1996) is adopted here to device neuro-fuzzy model. As the name suggests, an adaptive system is a network structure in which nodes are connected through directional links. These nodes may be partly or fully adaptive in nature, and their outputs are related to the

parameter(s) associated to these nodes. Hybrid learning rule postulates the method for the minimisation of prescribed error measure using these parameters. All 15 cost drivers and their six qualitative rating levels of COCOMO have been used in the proposed neuro-fuzzy model. Here for each neuro-fuzzy sub model (NF_i), we denote CD_i as the i^{th} effort multiplier where i = 1, 2, ..., 15. Here, CDR_{ij} is represented as the j^{th} rating input value of the i^{th} cost driver CD_i of NF_i and revised multiplier value REM_i is the corresponding output.

Figure 5 Proposed neuro-fuzzy-based effort estimation approach

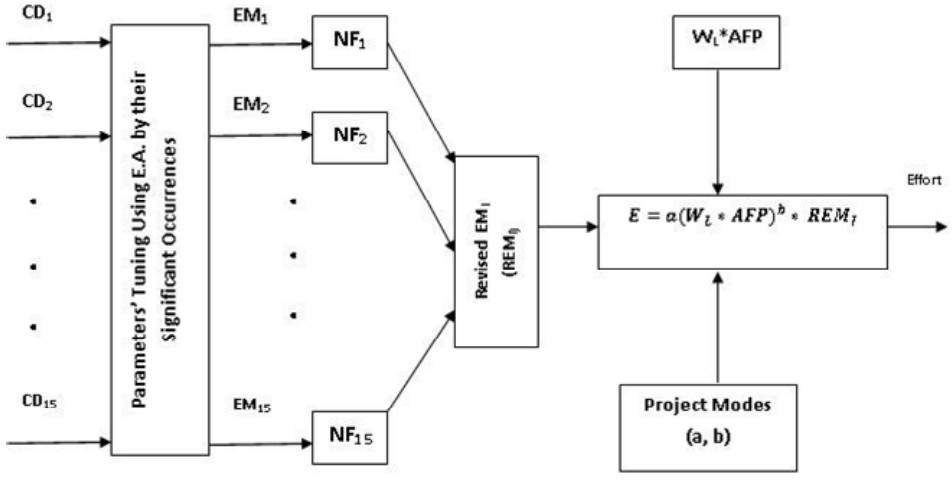
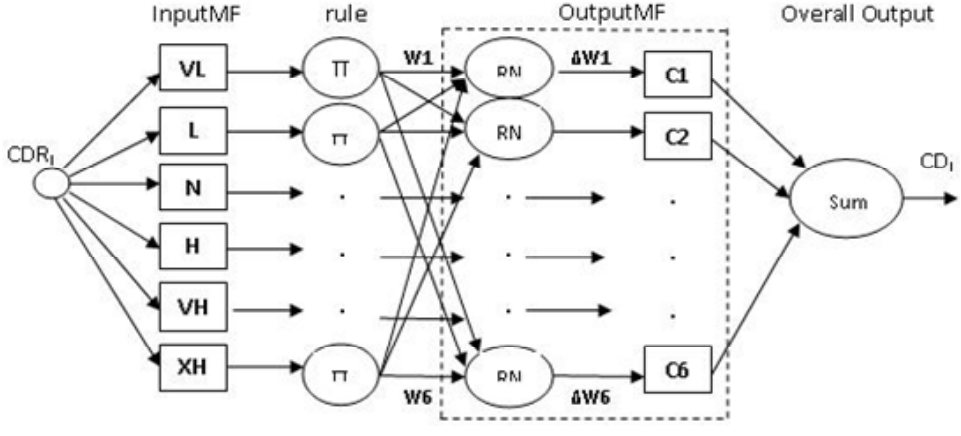


Figure 6 Sub-model (NF_i) structure of neuro-fuzzy approach



In Figure 6, 15 sub-models (NF_i) all together create a proposed approach structure whose functionality is similar to Takaki and Sugeno's (Jang, 1993) one-input – one-output type of fuzzy rule base system. This system contains simplified if-then fuzzy rules as described below by (1):

$$\text{If } CDR_i \text{ is } A_{ij}, \text{ then } f_j = CD_{ij},\ j = 1, 2, ..., 6 \text{ and } i = 1, ..., 15$$

where CDR_i is the linguistic variable and A_{ij} is the fuzzy set that defines the rating of linguistic terms like 'very low' to 'extra high'. Corresponding multiplier value of the cost driver CD_i is represented by CD_{ij}.

There are four functioning and processing levels named as inputMF, rule, outputMF and overall output. Functioning of each processing element of that level is of the same nature.

3.2.1 InputMF

At this level, membership function of A_{ij}^{th} processing element (rating values shown in rectangular boxes) states the point to which the certain input CDR_i fulfils the linguistic property variable A_{ij}. Generally, the membership function of CDR_i to be Gaussian curve built-in with maximum assumed to be 1 and minimum assumed to be 0 as stated below:

$$\mu A_{ij}\left(CDR_i\right) = \exp\left(-\frac{(x-a)^2}{2b^2}\right) \tag{6}$$

- *Rule:* at the next level of processing, the processing element labelled as $\prod$, performs the product of inputs to the perceptron to get the weighted output (W_{ij}) such as:

  $$W_{ij} = \mu A_{ij}\left(CDR_i\right) \tag{7}$$

  Here, $\prod$ shows that output is true if all inputs are true.

- *OutputMF:* in this processing unit, the elements named as RN represent a normalisation factor for the rules, which produce the weight corrections. It is the ratio of j^{th} weighted output to the sum of all k weighted outputs.

  $$\Delta W_{ij} = \frac{w_{ij}}{\sum_{k=1}^{6} W_k} \tag{8}$$

  where $j = 1, 2, 3, \ldots, 6$.

  Output membership function (O_j) of j^{th} element is the product of the element weight corrections and the consequent parameter (C_{ij}) is given by:

  $$O_j = \Delta W_{ij} C_{ij} \tag{9}$$

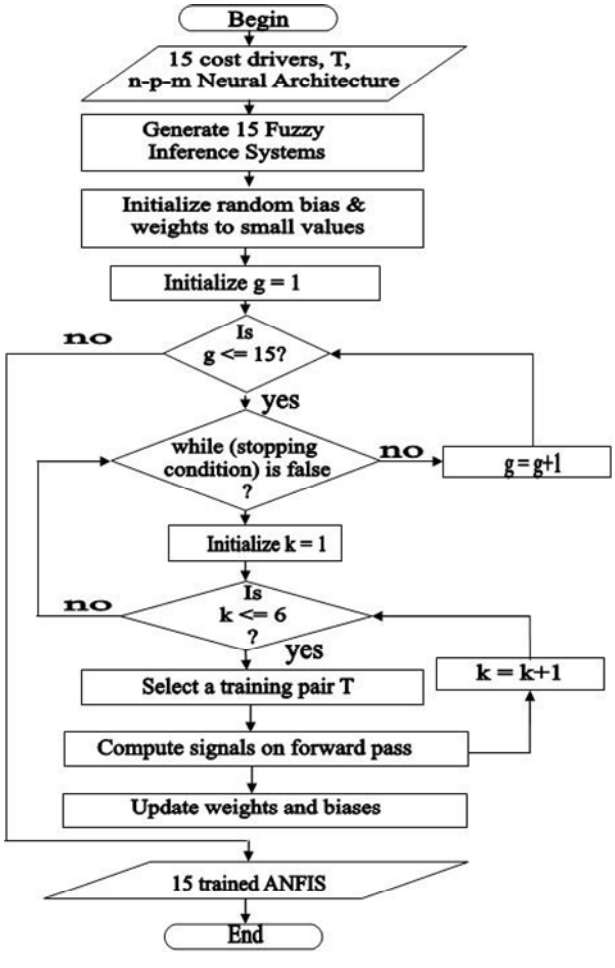
- *Overall output:* the circle node labelled as sum is computed by summing all output membership incoming signals, i.e.:

  $$Sum = \sum_{j=1}^{6} \Delta W_{ij} C_{ij} \tag{10}$$

3.3 Proposed hybrid learning algorithm

Before applying the proposed algorithm which is hybrid optimisation method, the ANFIS is trained in 16 steps by using least square method and back propagation gradient descent algorithm for the 15 cost drivers of COCOMO. The flow chart for training the ANFIS is given in Figure 7.

Figure 7 Flow chart for training the ANFIS



The parameters of proposed algorithm are as follows:

T training set for 15 cost drivers

X input training vectors

t output training vectors

R^n real valued inputs in the form of an n-dimensional vector

R^m real valued outputs in the form of an m-dimensional vector

n linear neurons

d number of project datasets

p hidden sigmoid neurons

m output sigmoid neurons

α momentum ($\alpha > 0$)

η learning rate

τ error tolerance

v biases

w weights

$\in$ pattern error

z hidden unit

Z activation function for hidden unit

y output unit

Y activation function for output unit

δ error information

ΔW bias correction and weights correction for output unit

ΔV bias correction and weights correction for hidden unit.

The stepwise details of the algorithm are as follows:

Given:

1 15 cost drivers with their values.

2 A training set T comprising vectors $X_{gk} \in R^n$.

3 Desired output vectors $t_{gk} \in R^m$.

4 *n-p-m* architecture neural network *N*.

where $g = 1 \ldots\ldots 15$, $k = 1 \ldots\ldots\ldots d$.

3.3.1 Generate the 15 fuzzy inference systems (FISs)

Step 1 Create a mapping between input and output membership functions along with the ratings of each cost driver. This mapping is defined by the fuzzy rules of Takagi and Sugeno (1983) type.

So we can define Fuzzy rules to develop 15 FISs.

Step 2 15 FISs (*FUZ*(15)) of Takagi and Sugeno type for corresponding 15 cost drivers are developed by following sequence of steps:

First take Gaussian input membership function for ratings of all 15 cost drivers.

Select constant as output membership function for ratings of all 15 cost drivers.

3.3.2 Initialise the FIS

Step 3 Set *FUZZ*(1:15, 1) = *FUZ*(15).

Set $mf(1) = gauss$, $mf(2) = tri$, $mf(3) = trap$.

Set $i = 1, j = 1$.

Step 4 Repeat Step 5 until ($i <= 15$).

Step 5 Repeat Steps 6 to 22 until ($j <= 3$).

3.3.3 Train the FIS by applying following steps

3.3.3.1 Initialisation of bias and weights

Step 6 Initialise random bias v_{ij} to small values, set $\Delta v_{ij}^0 = 0,\ i = 0, \ldots.., n; j = 1 \ldots.. p$.

Initialise random weight w_{jk} to small values, set $\Delta w_{jk}^0 = 0, j = 0, \ldots.., p$;

$k = 1 \ldots.. m$.

Set $k = 1$, α, η and the error tolerance τ as desired.

Step 7 For $g = 1$ to 15 repeat Steps 6–13.

Step 8 Repeat Steps 7–13 until $\varepsilon = \frac{1}{T}\sum_{k=1}^{T}\varepsilon_k < \tau$ is false.

Step 9 For every training pair $(X_{gk}, t_{gk}) \in T$. Do Steps 8–13.

3.3.3.2 Compute signals on forward pass in the following sequence

Step 10 At selected training pair (X_{gk}, t_{gk}), every input perceptron receives the input signal x_i and transfer these feeds to the adjacent layers with the hidden units.

Step 11 Every hidden unit ($z_j, j = 1, \ldots\ldots, p$) adjacent to input layer adds the product input with the weights as per the equation:

$$z_{-inj} = V_{oj} + \sum_{i=1}^{n} x_{gi} v_{ij} \tag{11}$$

After that, function $Z_j = f(z_{inj})$ is applied to all layers of output units.

where BINARY SIGMOIDAL function for hidden unit is given as:

$$f\left(z_{inj}\right) = \frac{1}{1 + e^{-\lambda z_{inj}}} \tag{12}$$

Step 12 Every output perceptron ($y_k, k = 1, \ldots\ldots.., m$) performs the summation of weighted input values it receives:

$$y_{-ink} = w_{ok} + \sum_{j=1}^{n} Z_j w_{jk} \tag{13}$$

And applies its BINARY SIGMOIDAL function generate final output value.

$$Y = f\left(y_{-ink}\right)$$

where BINARY SIGMOIDAL function for output unit is given as:

$$f\left(y_{-ink}\right)=\frac{1}{1+e^{-\lambda y_{-ink}}} \tag{14}$$

3.3.3.3 *Applying least square method for calculating the errors and back propagating the errors*

Step 13 The error in the produced output from output perceptron is estimated with the help of target and input pattern known to us.

$$\delta_k=\left(t_{gk}-y_k\right)f\left(y_{-ink}\right) \tag{15}$$

Step 14 Similarly the error information term for the input layer is estimated:

$$\delta_j=\delta_{-ink}f\left(z_{-inj}\right) \tag{16}$$

where delta inputs are calculated as:

$$\delta_{-ink}=\sum_{k=1}^{m}\delta_j w_{jk} \tag{17}$$

3.3.3.4 *Correction of weights and biases for updating FIS parameters*

Step 15 Correction in the weight is calculated as described below:

$$\Delta W_{jk}=\alpha\delta_k z_j \tag{18}$$

Similarly the bias correction at the input layer is described below:

$$\Delta W_{ok}=\alpha\delta_k \tag{19}$$

Consequently

$$\Delta W_{ok}(new)=\Delta W_{ok}(old)+\Delta W_{ok} \tag{20}$$

$$\Delta W_{jk}(new)=\Delta W_{jk}(old)+\Delta W_{jk} \tag{21}$$

Correction in the weight is described as:

$$\Delta V_{ij}=\alpha\delta_j x_{gi} \tag{22}$$

And change in bais at the input stage is described below:

$$\Delta V_{oj}=\alpha\delta_j \tag{23}$$

Consequently

$$\Delta V_{ij}(new)=\Delta V_{ij}(old)+\Delta V_{ij} \tag{24}$$

$$\Delta V_{oj}(new)=\Delta V_{oj}(old)+\Delta V_{oj} \tag{25}$$

Step 16 15 ANFIS will be generated as output.

3.3.4 Evaluate 15 ANFIS and update FIS

Step 17 *em*(1:15) = evaluate *FUZZ*(1:15, *j*).

Step 18 Calculate *MMRE*(*j*).

Step 19 Update membership function of *FUZZ*(*I*, *j*) = *mf*(*j*).

Step 20 If (*j* = = 3).

Step 21 Set *FUZZ*(*I*, *j*) to *mf*(*j*) where *j* corresponds to minimum *MMRE*[*j*].

Step 22 *i* = *i* + 1.

Step 23 Get the ANFIS with updated membership functions.

4 Evaluation criteria

For the performance the two parameters are selected they are *MMRE* and *PRED*(*L*). *MMRE* is the average of magnitude of relative error. *PRED* describes the ratio of project having relative error less than *L*.

$$MRE_i = \frac{\left|estimated\ effort_i - actual\ effort_i\right|}{actual\ effort} \tag{26}$$

As we have *N* project so we have to calculate the *MRE* of each project which implies that in the above equation $1 \le i \le N$. Finally the average of these values is the MMRE of the approach under test. The formula of MMRE as follows:

$$MMRE = \frac{1}{N}\sum_{i=1}^{N} MRE_i \tag{27}$$

Another parameter for evaluation is the prediction at level *L*, $PRED = \frac{k}{N}$.

To find the value of '*k*' we count all the number of projects for which the $MRE \le L$ and *N* is the total number of observations.

5 Result and discussion

5.1 Dataset description

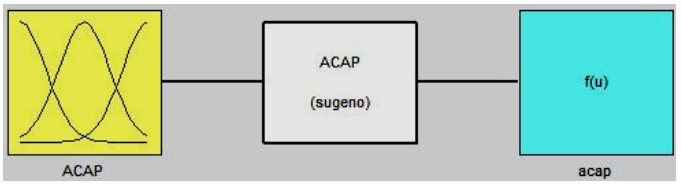
In the study data of NASA 93 projects is used. The dataset basically contains three development, one attribute and 15 cost drivers. As the projects are developed under observation there actual effort is also available which helps in the evaluation of the other approaches.

Figure 8 contains the inputs layer, outputs layer, and between them we have the central fuzzy rule processor. The whole process of FIS is described by following five steps:

Step 1: input

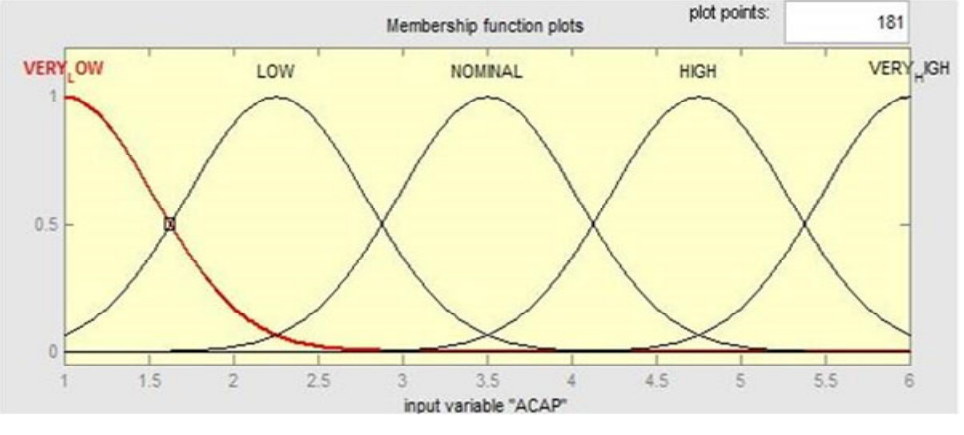
The first step involves crisp numerical input fuzzification. The input values are limited to the interval between 1 and 6. Output is generated as a degree of the appropriate fuzzy sets via membership functions in the qualifying linguistic set as the interval is always between 0 and 1. Input fuzzification deals with function evaluation.

Figure 8 Describing a single FIS (see online version for colours)



Our problem is built on 69 rules built on the different values of the cost drivers, and evaluation of each of the rules depends on inputs which must be fuzzified according to each fuzzy linguistic sets categorised as very low, low, nominal, high, very high and extra high. Figure 9 shows how efficiently the ACAP at rating scale of 1 to 6 qualifies, via its membership function, for linguistic variable very low. In this case, we rated ACAP as our graphical definition of nominal, corresponds to $\mu = 1$ for the nominal membership function.

Figure 9 Description of the Gaussian membership function for ACAP (see online version for colours)



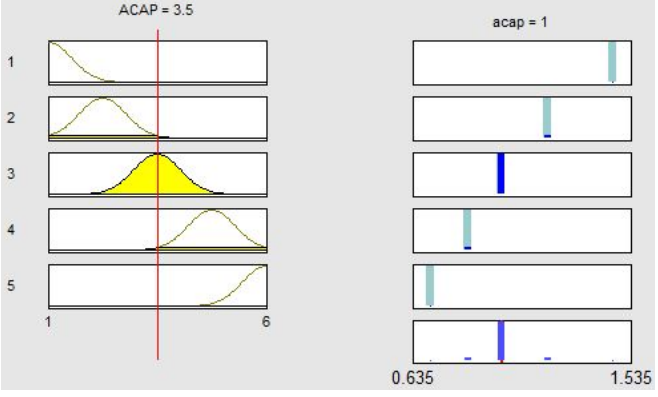
The same process is applied for fuzzification of each input over all the qualifying linguistic sets in membership functions which are required by the fuzzy rules.

Step 2: fuzzy operator application

As the crisp numerical input fuzzification process gets completed, the degree to which each part of the premise or antecedent satisfied for each and every rule will be known. Next, the fuzzy operator is applied in order to obtain one number that represents the

resultant antecedent for that rule. This piece of the antecedent of the rules for the ACAP calculation yielded the fuzzy membership 1, i.e., Figure 10, as ACAP is nominal.

Figure 10 Description of the rules of the fuzzy system for ACAP (see online version for colours)



As in the description of fuzzy logic operations, there are two well-defined built-in methods used for the AND operation and the OR operation. These methods are supported by prod (product) and probabilistic OR (probor) operations respectively. The probabilistic OR operation known as algebraic sum method is calculated using the following equation:

$$probor(a, b) = a + b - ab \tag{28}$$

Step 3: apply implication method

Every rule has a *weight* between 0 and 1, which is applied to the number given by the result of antecedent. Rule's weight must be determined, before applying the implication method. Generally, this weight is 1 which reflects the neutral effect on the implication process.

The implication method is implemented only after the proper weight has been assigned to each rule. A consequent is a linguistic characterised fuzzy membership function. The consequent shaping depends upon a function associated with the single number resultant antecedent. The implication process input is a single antecedent number, and the output is a fuzzy set. Same Implication process is implemented for each rule.

Step 4: aggregate all outputs

All rules must be placed together and tested in a FIS to make appropriate decisions. For that, process termed as aggregation, starts by combining output of each rule into a single fuzzy set. The membership function of every fuzzy set assigns a weight for every output value. Aggregation occurs for each output variable only once, just before the start of defuzzification step. The list of fuzzy sets for each rule is the input for the aggregation process which is returned by the implication process. The output of the aggregation

process is only one fuzzy set for each output variable. The order of rule execution is not that much important, as long as the aggregation process is commutative.

Step 5: defuzzify

In the process of defuzzification, the input is an aggregate output fuzzy set. Final desired output for each variable is represented by number as fuzziness helps the rule evaluation during the previous steps. However, fuzzy set aggregation encloses a range of the output values, and hence must be defuzzified so as to develop a single output value from that complete set of output values.

Conceivably, the centroid calculation is the most appraised defuzzification method, in this method we get the centre point of area bounded by the curve.

Figure 11 describes the adaptation of ACAP cost driver with the various iterations with 0 to 50.

Figure 11 Describing the error versus epochs during the training of ACAP cost driver (see online version for colours)

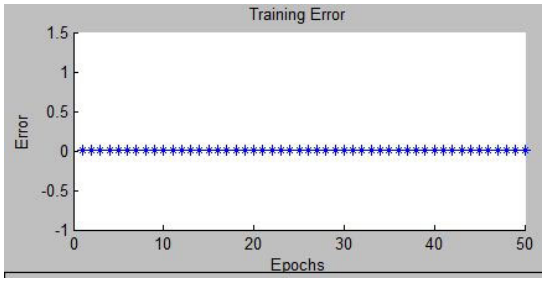


Figure 12 Describing the new membership function and rules in ANFIS (see online version for colours)

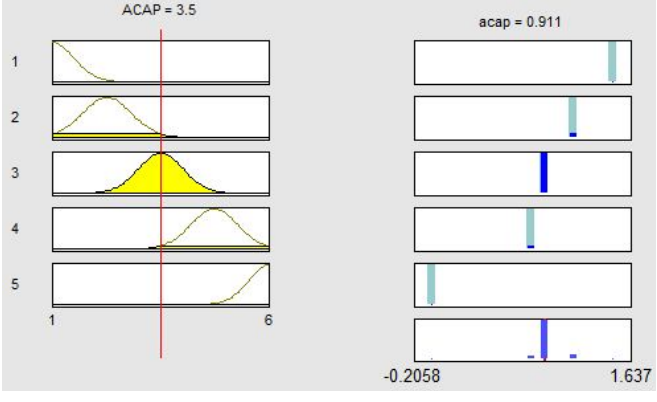


Figure 12 describes the membership function for ANFIS corresponding to the input value 3.5 and its effect on the output that is the combined effect of the three values shown by the dark blue lines.

The proposed model was validated by following two case studies.

Case 1

The results showing different MMREs in Table 1 obtained from different combinations of membership functions that are Gaussian, triangular and trapezoidal. In Table 1, initially we have taken the membership function for all the 15 cost drivers of Gaussian type. MMRE has been calculated for each cost driver. Now we replace the membership function for cost drivers by triangle membership function and then by the trapezoidal membership function. MMRE of all of the three are compared with each other and best membership is selected. This process is repeated for all the fifteen cost drivers. Finally we have achieved the MMRE improvement to 0.49 from 0.59.

Figure 13 Description of update in MMRE with the updation of membership of different cost drivers (see online version for colours)

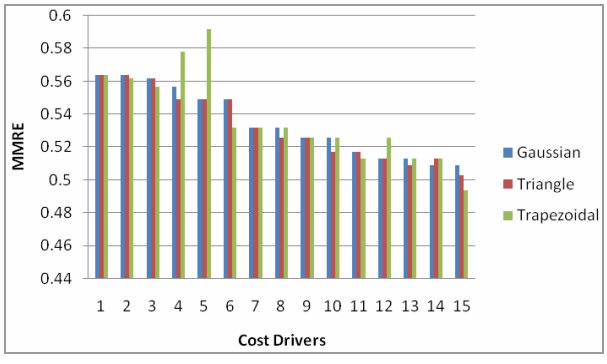


Table 1 Describing the updation in MMRE on the change of membership function

Sl. no.	*Cost driver*	*Gaussian*	*Triangle*	*Trapezoidal*
1	Rely	0.564	0.564	0.564
2	Data	0.564	0.564	0.562
3	Cplx	0.562	0.562	0.557
4	Time	0.557	0.549	0.578
5	Stor	0.549	0.549	0.592
6	Virt	0.549	0.549	0.532
7	Turn	0.532	0.532	0.532
8	Acap	0.532	0.526	0.532
9	Aexp	0.526	0.526	0.526
10	Pcap	0.526	0.517	0.526
11	Vexp	0.517	0.517	0.513
12	Lexp	0.513	0.513	0.526
13	Modp	0.513	0.509	0.513
14	Tool	0.509	0.513	0.513
15	Sced	0.509	0.503	0.494

The comparison of MMRE of all the three membership functions is described in Figure 13. From the figure, it is concluded that trapezoidal membership functions are not

suitable for the cost driver no. 4, 5, 12, i.e., Time, Stor, Lexp but the same membership function is good for the cost driver no. 11, i.e., Vexp. These are the major differences that can be inferred from the figure. Still some minor improvements in cost drivers can be observed from Table 1.

Figure 14 describes the improvement in MMRE by replacing the membership function from Gaussian to the best one from all the three, i.e., Gaussian, triangle and trapezoidal.

Figure 14 Figure describing MMRE achieved after the updation of cost drivers (see online version for colours)

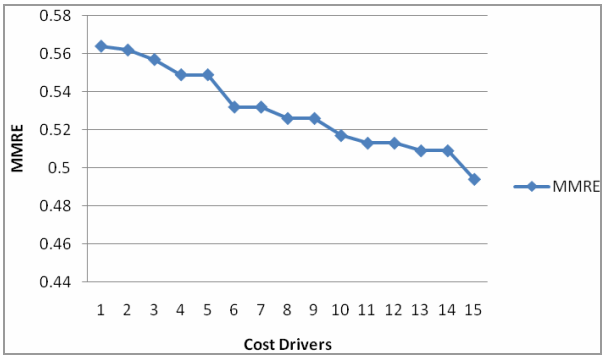


Table 2 Tuned parameter-based MMRE

MMRE (for 93 datasets)	
COCOMO vs. actual	*Tuned cost drivers*
0.6	0.56

Table 3 Describing the best membership function for each cost driver

Sl. no.	*Cost driver*	*Membership function*
1	Rely	Gaussian
2	Data	Trapezoidal
3	Cplx	Trapezoidal
4	Time	Triangular
5	Stor	Triangular
6	Virt	Trapezoidal
7	Turn	Gaussian
8	Acap	Triangular
9	Aexp	Gaussian
10	Pcap	Triangular
11	Vexp	Trapezoidal
12	Lexp	Triangular
13	Modp	Triangular
14	Tool	Gaussian
15	Sced	Trapezoidal

The modified cost drivers are used as the knowledgebase for the development of inference rules (Takagi and Sugeno, 1983). The MMRE for COCOMO cost drivers is 0.60 and if these cost drivers are tuned by evolutionary algorithm than it changes to 0.56 as in Table 2. The productivity measure is showing the difference in two methods by improving productivity significantly around 18% (Singh et al., 2013).

It can be observed by Table 3, that different membership functions are used for 15 cost drivers. The average of MMRE of Gaussian, triangular and trapezoidal membership function for all cost drivers are 0.535, 0.533, 0.537 respectively. So, Gaussian and triangular membership functions are producing a bit lesser average of MMRE than trapezoidal. Ten cost drivers are those who have their membership functions as Gaussian and triangular and rest of the cost drivers are using trapezoidal function.

Case 2

The results showing different MMREs in Table 4 obtained from different combinations of membership functions such as Gaussian, triangular and trapezoidal.

Table 4 MMRE for three different membership functions and their combinations

Membership function	*Gaussian*	*Trapezoidal*	*Triangular*	*Gau + Trap*	*Gau + Tri*	*Gau + Tri + Trap*
MMRE	0.593	0.594	0.595	0.538	0.593	0.534

In Table 4, MMRE has been calculated by first taking each membership function for all cost drivers individually, and then a combination of them. Results indicate that combination results in better performance of the system.

Figure 15 Relationship between MRE for neuro-fuzzy and COCOMO (see online version for colours)

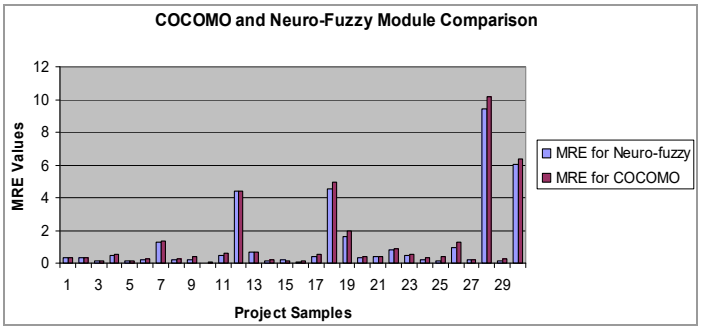


Table 5 Accuracy and improvement of the proposed method

Model	*Evaluation*	*MMRE*
Proposed approach vs. COCOMO'81	COCOMO'81	1.28
	Proposed approach	1.17
	Improvement %	10% (approx.)

Table 6 Comparison of predicted effort and actual effort for various methods

Model no.	*Estimation model/method*	*Average efforts*	*% deviation with actual value*	*% difference (comparison of 2, 4, 5, 6, 7 and 8 with 3)*
1	Actual	624.41		
2	COCOMO	809.07	29.57	28.92
3	proposed	627.56	0.50	
4	W-F	307.54	103.03	104.06
5	B&B	684.12	9.56	9.01
6	P, P&C	722.96	13.63	15.20
7	SEL	96.54	546.77	550.04
8	Nonlinear regression model	698.59	11.88	11.32

A comparison is made between proposed method and other estimation methods by MMRE in Figure 15. For approximately 30% randomly selected sample dataset, MMRE for COCOMO is 1.28 and for neuro-fuzzy approach is 1.17 which is shown in performance evaluation Table 5. Proposed approach is working efficiently for 93 datasets. Estimation accuracy is improved by approximately 10% for our method.

Table 6 depicts the comparisons of various approaches. COCOMO has the 29.57 percentage of deviation from actual dataset. Walston and Felix (1977) method has a large percentage of deviation of 103 percentage that is almost double of actual. Bailey and Basili (1981) model produces 9.56 percentage deviations from actual. P, P&C (Jodpimai et al., 2010) is producing approximately 14 percentage of difference from actual. SEL shows the higher side of differences with actual efforts. We have calculated the efforts with nonlinear regression model which shows the 11.88% deviation. It is observed that proposed method is always proved to be better as it is showing the minimum deviation of average efforts in comparison of other methods.

Table 7 Relative error comparison for various methods

Method	*MMRE*	*% improvement*
COCOMO	0.59	21.42
Proposed	0.49	
Walston Felix	0.60	21.96
Bailey and Basili	0.65	32.61
P, P&C	1.00	104.08
SEL	0.74	51.51
Nonlinear regression modelling (effort = 6.98 * (KLOC) ^ 0.92)	0.63	29.31

From experimental results it is observed that the MMRE of proposed software effort estimation approach was quite low than MMRE applying by other models and performs better estimation accuracy Table 7. The proposed approach, when applied to even larger size project, also yields better results. All these comparisons made one thing very clear that this combined approach is producing better results for almost every case of estimation.

6 Conclusions

By the research observations, it has been found that neuro-fuzzy approach works far better than COCOMO'81. Our work differs with those of others in the sense that in our case, three membership functions in fuzzy technique are used for validation of model with 93 software projects from NASA.

Our proposed method is a realistic method for providing handling the ambiguity present in software effort drivers. By combining the neuro-fuzzy method with the integrated approach of size estimation, we can find the better learning capability and well understood by using the fuzzy rules. It has been shown that it can greatly improve the performance of pre-existing model by removing a certain amount of error when compared with the standard COCOMO and other estimation methods. Therefore, for a randomly selected sample dataset and for complete dataset, error has been reduced or MMRE has been improved by approximately 10.104% in comparison with the other models.

Although the neural network approach can model complex relationship based on historical data, but it has a shortcoming: that it are tough to recognise and describe the judgment. The only parameters to be learned are EM_i. Locally in our decoupled learning approach. However, our neuro-fuzzy method is easy to understand how the decisions are taken and the parameters can be easily understood and validated by experts. Neuro-fuzzy method is suitable for project management. The efficient use of information from various available sources is done during decision making process by the neuro-fuzzy method. It is done by integrating expert knowledge of cost drivers and numerical project data using fuzzy rules to ensure that the tuned results are precise and realistic than conventional approaches. In future, we would like to add the optimisation algorithms to enhance the result of extended COCOMO.

References

Albrecht, A.J. (1979) 'Measuring application development productivity', in *Proc. Joint Share, Guide, and IBM Application Development Symposium*.

Albrecht, A.J. and Gaffney, J.E. (1983) 'Software function, source lines of code, and development effort prediction: a software science validation', *IEEE Transactions on Software Engineering*, Vol. SE-9, No. 6, pp.639–648.

Bailey, J.W. and Basili, V.R. (1981) 'A meta-model for software development resource expenditures', in *Proceedings of the 5th International Conference on Software Engineering*, pp.107–116.

Boehm, B., Abts, C. and Chulani, S. (2000) 'Software development cost estimation approaches – a survey', *Annals of Software Engineering*, Vol. 10, Nos. 1–4, pp.177–205.

Boehm, B.W. (2017) 'Software cost estimation meets software diversity', in *2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C)*, pp.495–496 [online] https://doi.org/10.1109/ICSE-C.2017.159.

Boehm, B.W. et al. (1981) *Software Engineering Economics*, Vol. 197, Prentice-hall Englewood Cliffs, NJ.

Briand, L.C., Basili, V.R. and Thomas, W.M. (1992) 'A pattern recognition approach for software engineering data analysis', *IEEE Transactions on Software Engineering*, Vol. 18, No. 11, pp.931–942.

Dote, Y. and Ovaska, S.J. (2001) 'Industrial applications of soft computing: a review', *Proceedings of the IEEE*, Vol. 89, No. 9, pp.1243–1265.

Du, W.L., Capretz, L.F., Nassif, A.B. and Ho, D. (2015) *A Hybrid Intelligent Model for Software Cost Estimation*, ArXiv Preprint ArXiv: 1512.00306.

Finnie, G.R., Wittig, G.E. and Desharnais, J-M. (1997) 'A comparison of software effort estimation techniques: using function points with neural networks, case-based reasoning and regression models', *Journal of Systems and Software*, Vol. 39, No. 3, pp.281–289.

Garratt P.W. and Hodgkinson A.C. (1999) 'A neurofuzzy cost estimator', *Proceedings of International Conference on Software Engineering and Applications*, pp.401–406.

Ghatasheh, N., Faris, H., Aljarah, I. and Al-Sayyed, R.M.H. (2019) *Optimizing Software Effort Estimation Models Using Firefly Algorithm*, ArXiv Preprint ArXiv: 1903.02079.

Gray, A.R. and MacDonell, S.G. (1997) 'A comparison of techniques for developing predictive models of software metrics', *Information and Software Technology*, Vol. 39, No. 6, pp.425–437.

Heemstra, F.J. (1992) 'Software cost estimation', *Information and Software Technology*, Vol. 34, No. 10, pp.627–639.

Idri, A., Zahi, A., Mendes, E. and Zakrani, A. (2007) 'Software cost estimation models using radial basis function neural networks', in *Software Process and Product Measurement, Lecture Notes in Computer Science*, Springer-Verlag, Vol. 4895, pp.21–31.

Jang, J-S. (1993) 'ANFIS: adaptive-network-based fuzzy inference system', *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. 23, No. 3, pp.665–685.

Jensen, R. (1983) 'An improved macrolevel software development resource estimation model', in *5th ISPA Conference*, pp.88–92.

Jodpimai, P., Sophatsathit, P. and Lursinsap, C. (2010) 'Estimating software effort with minimum features using neural functional approximation', in *2010 International Conference on Computational Science and Its Applications*, pp.266–273.

Jorgensen, M. (1995) 'Experience with the accuracy of software maintenance task effort prediction models', *IEEE Transactions on Software Engineering*, Vol. 21, No. 8, pp.674–681.

Kan, S.H. (2002) *Metrics and Models in Software Quality Engineering*, Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA.

Kemerer, C.F. (1987) 'An empirical validation of software cost estimation models', *Communications of the ACM*, Vol. 30, No. 5, pp.416–429.

Kitchenham, B.A. and Taylor, N.R. (1985) 'Software project development cost estimation', *Journal of Systems and Software*, Vol. 5, No. 4, pp.267–278.

Kumar, S., Krishna, B.A. and Satsangi, P.S. (1994) 'Fuzzy systems and neural networks in software engineering project management', *Applied Intelligence*, Vol. 4, No. 1, pp.31–52.

Li, Z. (2010) 'Intelligently predict project effort by reduced models based on multiple regressions and genetic algorithms with neural networks', in *2010 International Conference on E-Business and E-Government*, pp.1536–1542.

MacDonell, S.G. and Gray, A.R. (1998) 'A comparison of modeling techniques for software development effort prediction', *Proceedings of the International Conference on Neural Information Processing and Intelligent Information Systems*, Springer-Verlag, pp.869–872.

Malathi, S. and Sridhar, S. (2011) *A Classical Fuzzy Approach for Software Effort Estimation on Machine Learning Technique*, ArXiv Preprint ArXiv: 1112.3877.

Matson, J.E., Barrett, B.E. and Mellichamp, J.M. (1994) 'Software development cost estimation using function points', *IEEE Transactions on Software Engineering*, Vol. 20, No. 4, pp.275–287.

Mitra, S. and Hayashi, Y. (2000) 'Neuro-fuzzy rule generation: survey in soft computing framework', *IEEE Transactions on Neural Networks*, Vol. 11, No. 3, pp.748–768.

Mukhopadhyay, T. and Kekre, S. (1992) 'Software effort models for early estimation of process control applications', *IEEE Transactions on Software Engineering*, Vol. 18, No. 10, pp.915–924.

Mustapha, H., Abdelwahed, N. et al. (2019) 'Investigating the use of random forest in software effort estimation', *Procedia Computer Science*, Vol. 148, pp.343–352.

Pillai, K. and Nair, V.S.S. (1997) 'A model for software development effort and cost estimation', *IEEE Transactions on Software Engineering*, Vol. 23, No. 8, pp.485–497.

Putnam, L.H. (1978) 'A general empirical solution to the macro software sizing and estimating problem', *IEEE Transactions on Software Engineering*, Vol. 4, No. 4, pp.345–361.

Sandhu, P.S., Bassi, P. and Brar, A.S. (2008) 'Software effort estimation using soft computing techniques', *World Academy of Science, Engineering and Technology*, Vol. 46, pp.488–491.

Saxena, U.R. and Singh, S.P. (2012) 'Software effort estimation using neuro-fuzzy approach', in *2012 CSI Sixth International Conference on Software Engineering (CONSEG)*, pp.1–6.

Shepperd, M. and Kadoda, G. (2001) 'Comparing software prediction techniques using simulation', *IEEE Transactions on Software Engineering*, Vol. 27, No. 11, pp.1014–1022.

Shepperd, M. and Schofield, C. (1997) 'Estimating software project effort using analogies', *IEEE Transactions on Software Engineering*, Vol. 23, No. 11, pp.736–743.

Sheta, A., Rine, D. and Ayesh, A. (2008) 'Development of software effort and schedule estimation models using soft computing techniques', in *2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence)*, pp.1283–1289.

Shi, Y., Mizumoto, M., Yubazaki, N. and Otani, M. (1996) 'A learning algorithm for tuning fuzzy rules based on the gradient descent method', in *Proceedings of IEEE 5th International Fuzzy Systems*, Vol. 1, pp.55–61.

Singh, B.K., Tiwari, S., Mishra, K.K. and Misra, A.K. (2013) 'Tuning of cost drivers by significance occurrences and their calibration with novel software effort estimation method', *Advances in Software Engineering*, Vol. 2013, pp.1–10, Article ID 351913.

Srinivasan, K. and Fisher, D. (1995) 'Machine learning approaches to estimating software development effort', *IEEE Transactions on Software Engineering*, Vol. 21, No. 2, pp.126–137.

Takagi, T. and Sugeno, M. (1983) 'Derivation of fuzzy control rules from human operator's control actions', *IFAC Proceedings*, Vol. 16, No. 13, pp.55–60.

Venkatachalam, A.R. (1993) 'Software cost estimation using artificial neural networks', in *Proceedings of 1993 International Conference on Neural Networks*, IJCNN-93-Nagoya, Japan, Vol. 1, pp.987–990.

Venkataiah, V., Mohanty, R. and Nagaratna, M. (2019) 'Prediction of software cost estimation using spiking neural networks', in *Smart Intelligent Computing and Applications*, pp.101–112, Springer, Singapore.

Vijayakumar, S. (1997) 'Use of historical data in software cost estimation', *Computing & Control Engineering Journal*, Vol. 8, No. 3, pp.113–119.

Walston, C.E. and Felix, C.P. (1977) 'A method of programming measurement and estimation', *IBM Systems Journal*, Vol. 16, No. 1, pp.54–73.

Yu, W.D. (1990) 'A modeling approach to software cost estimation', *IEEE Journal on Selected Areas in Communications*, Vol. 8, No. 2, pp.309–314 [online] https://doi.org/10.1109/49.46886.

Zadeh, L.A. (1965) 'Fuzzy sets', *Information and Control*, Vol. 8, No. 3, pp.338–353.

Zadeh, L.A. (1994) 'Fuzzy logic, neural networks, and soft computing', *Communications of the ACM*, Vol. 37, No. 3, pp.77–85.