Hussain Azath*, Marimuthu Mohanapriya and Somasundaram Rajalakshmi

Software Effort Estimation Using Modified Fuzzy C Means Clustering and Hybrid ABC-MCS Optimization in Neural Network

https://doi.org/10.1515/jisys-2017-0121
Received March 21, 2017; previously published online February 1, 2018.

Abstract: In a software development process, effective cost estimation is the most challenging activity. Software effort estimation is a crucial part of cost estimation. Management cautiously considers the efforts and benefits of software before committing the required resources to that project or order for a contract. Unfortunately, it is difficult to measure such preliminary estimation, as it has only little information about the project at an early stage. In this paper, a new approach is proposed; this is based on reasoning by the soft computing approach to calculate the effort estimation of the software. In this approach, rules are generated based on the input dataset. These rules are then clustered for better estimation. In our proposed method, we use modified fuzzy C means for clustering the dataset. Once the clustering is done, various rules are obtained and these rules are given as the input to the neural network. Here, we modify the neural network by incorporating optimization algorithms. The optimization algorithms employed here are the artificial bee colony (ABC), modified cuckoo search (MCS), and hybrid ABC-MCS algorithms. Hence, we obtain three optimized sets of rules that are used for the effort estimation process. The performance of our proposed method is investigated using parameters such as the mean absolute relative error and mean magnitude of relative error.

Keywords: Cost estimation, neural network, clustering, fuzzy.

1 Introduction

With the popularization of information engineering, customers gradually realize the importance of project management and control in software projects. This requires accurate enough effort estimate and quality guarantee, which are provided by software developers [5]. Accurately predicting the software development effort is a critical concern of many organizations even today. Underestimating the development cost and schedule can have a detrimental impact on both the functionality and the quality of software products and therefore on the developer's reputation and competitiveness. Software project cost and effort estimation is an increasingly important field due to the overwhelming role of software in today's global market. However, there is no optimal approach to accurately predict the effort needed for developing a software system. Usually, the information gathered at the early stages of software system development is insufficient for providing a precise effort prediction [8]. Effort estimation methods could be based on subjective expert judgment or formal estimation models. Formal models use functional dependency to estimate effort using some value that quantifies the project size [15].

To manage a software project effectively, accurate and reliable software effort estimation is a critical activity. In the last decade, a number of software effort estimation methods with different concepts and approaches combined with existing effort estimation methods have been developed and evaluated using

***Corresponding author: Hussain Azath,** Research Scholar, CSE, Karpagam University, Coimbatore, India; and Assistant Professor, CSE, KGiSL Institute of Technology, Coimbatore, India, e-mail: writetoazath@yahoo.com
Marimuthu Mohanapriya: Research Supervisor, Department of CSE, Karpagam University, Coimbatore, India; and Associate Professor/Department of CSE, CIT, Coimbatore, India
Somasundaram Rajalakshmi: Professor & Head, Department of CSE, Cheran College of Engineering, Karur, India

historical project data [18]. A good estimation of the size and effort available right from the start in a project gives the project manager confidence about any future course of action, as many of the decisions made during development depend on, or are influenced by, the initial estimations. Hence, effort estimation is one of the most crucial steps of planning and management of a software project [1]. Development effort is considered to be one of the major components of software costs, particularly as regards global development, and it is usually the most challenging effort to predict. Several methods with which to support project managers when estimating software development effort for global software development projects have been proposed in the last few years [7].

To provide precise estimates and decrease the estimation error, several cost estimation models have been proposed and developed. Among these models, the constructive cost model (COCOMO) is the most commonly used model in software companies because of its capabilities and features for estimating the software effort in person-month at different development stages [2]. In the COCOMO, the basic model is usually applied in the early phase of software development. When more detailed project information can be obtained in the later phase, the intermediate and advanced models can be used for project estimation and measurement [2, 12]. As far as effort estimation is concerned, a number of unsolved problems and errors still exist. Hence, to obtain good results, it is essential to consider data from previous projects. The need of accurate estimation has been always important while determining the feasibility of a project in terms of cost-benefit analysis [16].

The rest of the paper is organized as follows. Section 2 explains the researches related to our proposed method. Section 3 shows our proposed method being used for the software effort estimation process. Section 4 explains the result of our proposed method. Finally, Section 5 concludes our proposed method with suggestions given for future works.

2 Related Work

Software development effort is one of the most important metrics that must be correctly estimated in software projects. Analogy-based estimation (ABE) and artificial neural networks (ANNs) are the most popular methods used widely in this field. These methods suffer from inconsistent and irrelevant projects that exist in software project datasets. Bardsiri et al. [4] have proposed to increase the accuracy of development effort estimation based on the combination of fuzzy clustering, ABE, and ANN methods. In our proposed method, the effect of irrelevant and inconsistent projects on estimates is decreased by designing a framework in which all the projects are clustered. The quality of training in ANN and the consistency of historical data in ABE are improved using the proposed framework. Two large and real datasets are used to evaluate the performance of our proposed method and the obtained results are compared to eight other estimation methods.

A predictive model is required to be accurate and comprehensible to inspire confidence in a business setting. Dejaeger et al. [6] have addressed the issue by reporting the results of a large-scale benchmarking study. Different types of techniques are under consideration, including techniques inducing tree/rule-based models such as M5 and CART, linear models such as various types of linear regression, nonlinear models (MARS, multilayered perceptron neural networks, radial basis function networks, and least-squares support vector machines), and estimation techniques that do not explicitly induce a model (e.g. a case-based reasoning approach). Furthermore, the aspect of feature subset selection using a generic backward input selection wrapper is investigated. The results are subjected to rigorous statistical testing and indicate that ordinary least-squares regression in combination with a logarithmic transformation performs best. Another key finding is that by selecting a subset of highly predictive attributes such as project size, development, and environment-related attributes, typically a significant increase in estimation accuracy could be obtained.

ABE is one of the efficient methods for software effort estimation because of its outstanding performance and capability of handling noisy datasets. Conventional ABE models usually use the same number of

analogies for all projects in the datasets to make good estimates. Therefore, there is a need to better understand the dataset characteristics to discover the optimum set of analogies for each project rather than using a static k-nearest projects. Azzeh and Nassif [3] have proposed a technique based on bisecting k-medoids clustering algorithm to come up with the best set of analogies for each individual project before making the prediction. With bisecting k-medoids, it is possible to better understand the dataset characteristic and automatically find the best set of analogies for each test project.

Due to the increasing complexity of software development activities, the need for effective effort estimation techniques has arisen. Underestimation leads to the disruption in the project's estimated cost and delivery. On the contrary, overestimation causes outbidding and financial losses in business. Effective software effort estimation techniques enable project managers to schedule the software life-cycle activities appropriately (Wijayasiriwardhane et al. [19]). Correctly assessing the effort needed to develop a software product is a major concern in software industries. Random forest (RF) technique is a popularly used machine learning technique that helps in improving the prediction values. Satapathy et al. [17] have precisely assessed the software project development effort using the case-point approach. The effort parameters are optimized using the RF technique to acquire higher prediction accuracy. Moreover, the results acquired by applying the RF technique are compared to multilayer perceptron, radial basis function network, stochastic gradient boosting, and log-linear regression techniques to highlight the performance attained by each technique.

To characterize the essential content of the SEE data, the least number of features and instances is required to capture the information within the SEE data. If the essential content is very small, then (1) the contained information must be very brief and (2) the value added of complex learning schemes must be minimal. Kocaguneli et al. [11] have computed the Euclidean distance between rows (instances) and columns (features) of SEE data, pruned synonyms (similar features) and outliers (distant instances), and then assessed the reduced data by comparing predictions from (1) a simple learner using the reduced data and (2) a state-of-the-art learner (CART) using all data.

2.1 Limitations

- The understanding and calculation of models based on historical data is difficult due to the inherent complex relationships between the related attributes that are unable to handle categorical data as well as the lack of reasoning capabilities.
- Various software design methods and processes address the issue of predicting development times.
- However, the limited accuracy of these cost models is not the fundamental obstacle to software estimation. Rather, as cost models are a function of a size or complexity measure, the issue is how to estimate the size or complexity of a new project.

3 Proposed Methodology for Software Effort Estimation

3.1 Steps Involved in Our Proposed Scheme of Effort Estimation

In this paper, a new approach is proposed; this is based on reasoning by the soft computing approach to calculate the effort estimation of the software. In this approach, first the datasets are clustered to create the rules. In our proposed method, we use modified fuzzy C means (MFCM) for clustering the dataset. Once the clustering is done, various rules are obtained and these rules are given as the input to the neural network. Here, we modify the neural network by incorporating optimization algorithms. The optimization algorithms employed here are the artificial bee colony (ABC), modified cuckoo search (MCS), and hybrid ABC-MCS algorithms. Hence, we obtain three optimized sets of rules that are used for the effort estimation process. The overall flow diagram of our proposed method is shown in Figure 1.

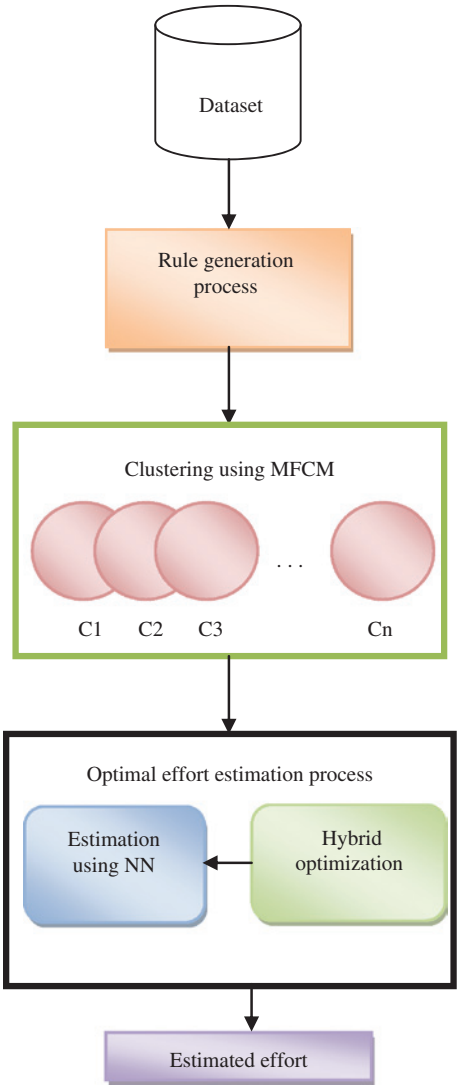

Figure 1: Proposed Software Effort Estimation Model.

3.2 Rules Generation Process

In our proposed method, we use four datasets: Desharnais, COCOMO 81, NASA 60, and NASA 93. Here, we have the Desharnais dataset as the input. The dataset usually contains high and low effort projects. Based on the attributes and properties given for each project, we have to estimate the efforts. For better accuracy, here we generate the rules for the values given in the dataset.

Initially, the rules are generated from the input parameters. The given dataset contains numerical values. To generate the rules, we have to do the normalization process and convert them into a linguistic variable such as high, medium, and low. Based on these variables, the rules will be generated as given in Table 1.

3.3 Clustering of Rules Using the MFCM Algorithm

The MFCM algorithm is commonly used for clustering where the performance of the MFCM depends on the selection of the initial cluster center or membership value. The MFCM algorithm starts with a set of initial cluster centers (or) arbitrary membership values.

Table 1: Rules Generated from the Input Variables.

Team exp	Manager exp	Length	Transaction	Entities	Point adjust	Envergure	Language
L	L		M	L	H	L	H
L	M	L	H	L	L		L
M		M	L		L	H	M
H	L	L	H	H		H	H

The MFCM algorithm assigns pixels to each category using fuzzy memberships.

$$M = \sum_{i=1}^{I}\sum_{j=1}^{J}(\mu_{ij})^{v}\frac{\|t_i - c_j\|^2}{\rho_i} \tag{1}$$

where t_i is the extracted testcase, c_j is the cluster center, v is the constant value, and ρ_i is the weighted mean distance in cluster i and is given by

$$\rho_i = \left\{\frac{\sum_{j=1}^{n}\eta_{ij}^{v}\|t_i - c_j\|^2}{\sum_{j=1}^{n}\eta_{ij}^{v}}\right\}^{1/2} \tag{2}$$

The membership function represents the probability that a pixel belongs to a specific cluster. In the FCM algorithm, the probability is dependent on the distance between the pixel and each individual cluster center in the feature domain. The membership functions and cluster centers are updated by Equations (3) and (4).

$$\eta_{ij} = \frac{1}{\sum_{k=1}^{J}\left(\frac{\left\|\frac{t_i - c_j}{\rho_i}\right\|}{\left\|\frac{t_i - c_k}{\rho_i}\right\|}\right)^{\frac{2}{v-1}}} \tag{3}$$

The cluster centroid values are computed using Equation (4):

$$c_j = \frac{\sum_{i=1}^{I}\eta_{ij}^{v}\cdot t_i}{\sum_{i=1}^{I}\eta_{ij}^{v}} \tag{4}$$

Repeat the algorithm until the coefficients change between two iterations for the given sensitivity threshold.

$$\max_{ij}\|\eta_{ij}^{(k)} - \eta_{ij}^{(k+1)}\| < \varphi \tag{5}$$

In Equation (5), φ is a termination criterion between 0 and 1. Repeat the process until efficient clustering is reached. Thus, from the MFCM, the rules are clustered.

3.4 Rules Selection Using the Hybrid Neural Network (HNN) with ABC-MCS

The HNN is used to ascertain the rules generated and it is trained by employing the various rules employed in the proposed scheme of effort estimation. The hybrid ANN is well trained by means of the extorted rules. The innovative HNN is home to three input units, n hidden units, and one output unit. The input of the neural network are the rules we have generated. The neural network works by making use of two phases: one is the training phase and the other is the testing phase.

(A) Training Phase: In the training phase, the rules are given as the input to the neural network. Initially, the nodes are given random weights. As the output is already known in the training phase, the output obtained from the neural network is compared to the original, and weights are varied to reduce the error. This process is carried for every rule to yield a stable system having weights assigned in the nodes.

(B) Testing Phase: In the testing phase, the rules are fed to the trained neural network having particular weights in the nodes and the output is calculated based on the trained dataset. In the ordinary neural network, the process will be stopped after testing. In the proposed HNN, for the testing process, we incorporate the optimization algorithm to optimize the weight used for testing. In our proposed method, the weights are optimized with the help of the ABC-MCS Algorithm. By incorporating the optimization process, the selection accuracy will be improved by providing a better effort estimation process. The structure of the ANN is shown in Figure 2.

3.4.1 Proposed ABC-MCS for the Optimization of Weights in a Neural Network

In an ABC model, a food source position refers to a possible solution for the optimization problem, whereas the nectar quantity of a food source refers to the quality (fitness) of the respective solution. The objective of bees is set to determine the best solution [10]. Each employed bee shares the information with an onlooker bee and flies back to the food source, which was visited by it in the previous wandering. This process is undertaken as the memory keeps the record of the food source. The employed bee selects a new food source using the visual knowledge about the neighborhood of the one stored in the memory and assesses the nectar amount [9].

3.4.1.1 Employee Bee Phase

There are three bee groups in the ABC: employed bees, onlooker bees, and scout bees. An employed bee is a bee that flies back to its previously visited food source, whereas an onlooker bee is a bee that resides in the dancing area to decide on the food source to be selected. Scout bees are bees that undergo random wandering for food source. Employed bees contribute the first half of the colony, whereas the rest is contributed by onlooker bees. Each employed bee contributes by a food source. In other words, the number of food sources around the hive and the number of employed bees are same.

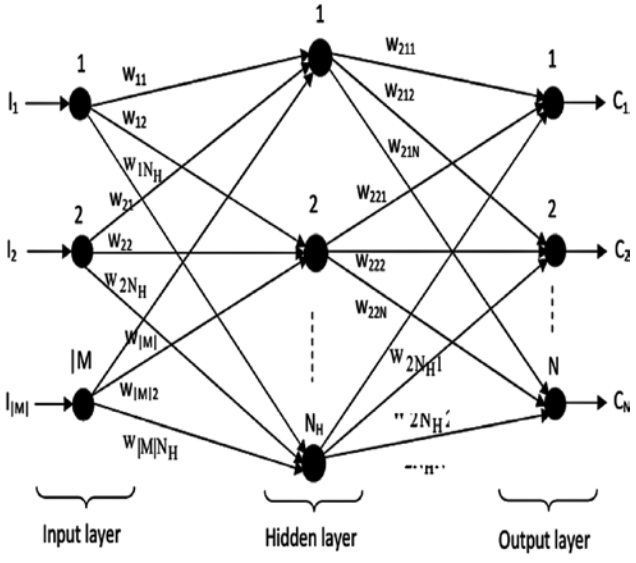

Figure 2: Structure of the ANN.

Initially, employed bees select a random set of food source positions for which the nectar amounts are calculated. They reach the hive and distribute the nectar information of the food sources to onlooker bees, which are in the dancing region of the hive. In the perspective of the algorithm, the arbitrary set of initial population p_{inl} with m solutions, where each solution is the food source position and P_{sz} is the population size that is generated. The solution representation can be given as S_i, where $1 \le i \le m$ is an N-dimensional vector and M is the number of parameters to be optimized. Once the population is initialized, it is subjected to the iterative process that involves employed, onlooker, and scout bees.

3.4.1.2 Onlooker Bee Phase

This phase enables onlooker bees to select the food sources based on the nectar information obtained from employed bees and to generate new set of solutions. Generally, the onlooker bee is font of the food source area, which has substantial nectar information shared by employed bees in the dancing region of the hive. The probability of selecting a food source by an onlooker bee is directly proportional to the nectar amount of the food source. Hence, an employed bee dancing with higher nectar value assigns an onlooker bee for the food position. The probability of selecting a food source (f_{sou}) by an onlooker bee can be given as follows:

$$f_{sou} = \frac{f_{\text{sln}}}{\sum_{i=1}^{m} f_i} \tag{6}$$

where f_{sln} is the fitness of the solution and m is the number of food sources, which is equal to the number of employed bees.

The onlooker bee reaches the selected food source and selects a new neighborhood of the selected food source based on visual knowledge. The visual knowledge is obtained by comparing both food positions. When the bee abandons a food source due to its lesser nectar value, the scout bee generates a new random food source and fills the abandoned position. The onlooker bee modifies the food source position stored in the memory and hence finds new a food source and validates the nectar amount (fitness) of it.

If we consider an old position $O_{i,k}$ and a new position $N_{i,k}$, the relationship can be given as

$$N_{i,k} = O_{i,k} + \sigma_{i,k}(O_{i,k} - O_{j,k}),\ i \neq j \tag{7}$$

where $j = \{1, 2, \ldots, n\}$, $k = \{1, 2, \ldots, N\}$, and $\sigma_{i,k}$ is the arbitrary number in the range [–1,1].

The position update equation interprets that a decrease in the deviation between the parameters of $O_{i,k}$ and $O_{j,k}$ leads to a decrease in the perturbation on the position $O_{i,k}$. Hence, an adaptive reduction in the step length happens, when the optimal solution in the search space has reached. Reformulating the position updating step leads to the following equation:

$$N_{i,k} - O_{i,k} = \sigma_{i,k}(O_{i,k} - O_{j,k}) \tag{8}$$

A time domain representation can be given for the position update equation by considering $O_{i,k}$ as D_h when $N_{i,k}$ is taken as D_{h+1}. Hence, we obtain

$$D_{h+1} - D_h = \sigma_{i,k}(O_{i,k} - O_{j,k}) \tag{9}$$

As $D_{h+1} - D_h$ refers to discrete version of the derivative of order $\lambda = 1$, we can write

$$Q^{\lambda}[D_{h+1}] = \sigma_{i,k}(O_{i,k} - O_{j,k}) \tag{10}$$

The output from the onlooker bee phase is the applied to the MCS algorithm for further optimization, which can aid in better weight optimization. The cuckoo search algorithm represents a metaheuristic algorithm that owes its origin to the breeding conduct of the cuckoos, and it is easy to implement. There is a multitude of nests in the cuckoo search. Each egg signifies a solution and an egg of cuckoo corresponds to a novel solution. The novel and superior solution replaces the most horrible solution in the nest. Similar to the modified neural network, we also modify the ordinary cuckoo search algorithm by including the Gauss

distribution in the updation phase where levy flight equation is used. The Gauss distribution adds better results of optimization when compared to the normal process. The modus operandi of the clustering procedure is shown as follows:

Step 1: Initialization Phase
The population (m_i, where $i = 1, 2, n$) of the host nest is initiated arbitrarily.

Step 2: Generating New Cuckoo Phase
With the help of the levy flights, a cuckoo is selected randomly, which generates novel solutions. Subsequently, the engendered cuckoo is evaluated by employing the objective function for ascertaining the excellence of the solutions.

Step 3: Fitness Evaluation Phase
The fitness function is evaluated in accordance with Equations (11) and (12) followed by the selection of the best one:

$$P_{\max} = \frac{P_S}{P_T} \tag{11}$$

$$\text{Fitness} = \text{maximum popularity} = P_{\max} \tag{12}$$

where P_s is the selected population and P_T is the total population.

Step 4: Updation Phase
At the outset, the solution is optimized by the levy flights by employing the cosine transform. The quality of the novel solution is evaluated and a nest is selected arbitrarily from among them. If the quality of the novel solution in the selected nest is superior to the previous solution, it is replaced by the novel solution (cuckoo). Otherwise, the previous solution is treated as the best solution. The levy flights employed for the general cuckoo search algorithm is expressed by Equation (13):

$$m_i^* = m_i^{(t+1)} = m_i^{(t)} + \alpha \oplus \text{Levy}(n) \tag{13}$$

By suitably adapting Equation (13), the levy flight equation using the Gauss distribution is exhibited by Equation (14):

$$m_i^* = m_i^{(t+1)} = m_i^{(t)} + \alpha \oplus \sigma_s \tag{14}$$

where

$$\sigma_s = \sigma_0 \exp(-\mu\kappa) \tag{15}$$

σ_0, μ represents the constants and K symbolizes the current generation.

Step 5: Reject Worst Nest Phase
In this section, the worst nests are ignored in accordance with their possibility values and novel ones are constructed. Subsequently, depending on their fitness function, the best solutions are ranked. Thereafter, the best solutions are detected and marked as optimal solutions.

Step 6: Stopping Criterion Phase
Until the achievement of the maximum iteration, the procedure is continued. By deftly employing the above-mentioned optimization process, we obtain an improved rate of effective rules selection that aid in an improved effort estimation of the software.

4 Results and Discussion

In our proposed method, we employ the modified ABC-MCS algorithm for the rules optimization to find out software effort estimation. Our proposed method is implemented in Netbeans 7.4, which is a renowned platform to use for effort estimation process because of its compatibility. The platform offers a variety of applications that can be developed from a set of components, which are often referred to as modules. As Netbeans has a collective set of modules as built-in functions to develop a program, it is convenient for any user to initiate the work immediately. In our proposed method, we use four datasets: Desharnais, COCOMO 81, NASA 60, and NASA 93.

4.1 Dataset Description

The dataset exploited in the revision is NASA 60. The NASA 60 dataset includes 60 whole projects, with 17 autonomous variables of which 15 are unconditional.

The NASA 93 dataset includes 93 whole projects, with 17 autonomous variables of which 15 are unconditional.

The COCOMO 81 dataset comprises 81 projects with 17 attributes and 63 instances.

The Desharnais dataset has 81 projects with four incomplete projects that can be removed. There are nine independent variables and one dependent variable in the dataset.

The effort obtained using our proposed method is tabulated in the following table along with the actual effort. The MRE value for our proposed method is then calculated based on the effort estimated and the actual effort being calculated.

Although numerous error measures are in practice, the most renowned error measure is the mean absolute relative error (MARE).

$$\mathrm{MARE} = \sum_{\mathrm{eff}=1}^{n} (|(\mathrm{Est}_{\mathrm{eff}} - \mathrm{Act}_{\mathrm{eff}}) / \mathrm{Act}_{\mathrm{eff}}|) \tag{16}$$

where $\mathrm{Est}_{\mathrm{eff}}$ is the estimated effort and $\mathrm{Act}_{\mathrm{eff}}$ is the actual effort.

4.2 Experimental Results

The obtained experimental outcomes from our proposed method are tabulated in Table 1. The actual and estimated efforts are determined for various sizes followed by determining the magnitude of relative error (MRE) for each entry. The actual effort is typically lesser than the estimated effort. In Equation (18), the mean MREs (MMREs) for the efforts are determined for the execution time. Table 2 shows the estimated and actual efforts based on which the MRE value is estimated.

Table 2: Comparison of the Estimated and Actual Efforts.

Dataset	Project no.	Actual effort	Estimated effort
Desharnais	1	7538.342	7392.341
	2	5037.401	4989.401
	3	797.177	953.176
NASA 93	1	304.1023	345.76
	2	241.1755	173.45
	3	442.0866	351.8
NASA 60	1	121.625	117.55
	2	121.625	130.95
	3	122.333	138.34
COCOMO 81	1	2040	2033.56
	2	1600	1606.16
	3	243	235.01

4.3 Performance Analysis

The following equations are used to determine the MRE and MMRE.

$$\mathrm{MRE} = |(\mathrm{Act}_{\mathrm{eff}} - \mathrm{Est}_{\mathrm{eff}})| / \mathrm{Act}_{\mathrm{eff}} \tag{17}$$

where $\mathrm{Est}_{\mathrm{eff}}$ is the estimated effort and $\mathrm{Act}_{\mathrm{eff}}$ is the actual effort.

The MMRE calculation for the estimated effort can be done using Equation (18). Our proposed method has accomplished better MMRE than did other fuzzy-based works.

$$\mathrm{MMRE} = \frac{1}{n}\sum_{i=1}^{n} \mathrm{MRE}_i \tag{18}$$

The graphical representation of the obtained effort value from our proposed method is shown in Figure 3 in which the actual effort is relatively higher than the estimated effort values.

The accomplished improvement in effort estimation is observed using the MMRE and MARE of our proposed method before and after the optimization process. The values are measured in percentage. The obtained values are given in Table 3.

The performance improvement accomplished from the post-optimization process is better illustrated in Figure 4, where the error values obtained from before and after optimization are plotted.

The effectiveness of our proposed method is then proven by comparing it to existing methods through MMRE measurements obtained from proposed and existing methods. Table 4 shows the original efforts and estimated efforts by various existing effort estimation algorithms. Our proposed method estimated the effort more accurately than did the other algorithms. Because of the rule generation process and clustering of rules. Here, we group the projects into low and high efforts. These clustered projects are trained separately in the optimal neural network for better estimation. Here, the MMRE is given as the fitness function for optimization.

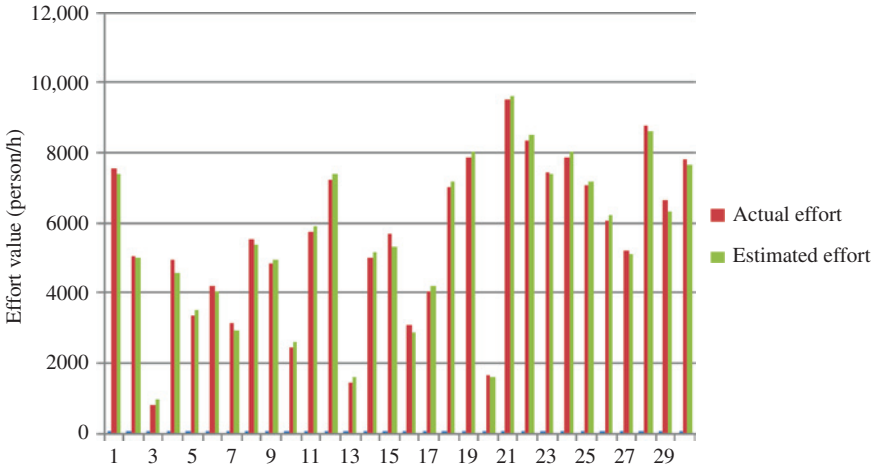

Figure 3: Graphical Representation of the Effort Values of the Desharnais Dataset.

Table 3: Parameter Values Before and After Applying the ABC Algorithm for the Optimization for the Desharnais Dataset.

Parameters	Proposed results (%)	
	Before optimization	After optimization
MMRE	13.608	0.04101
MARE	65.639	1.23027

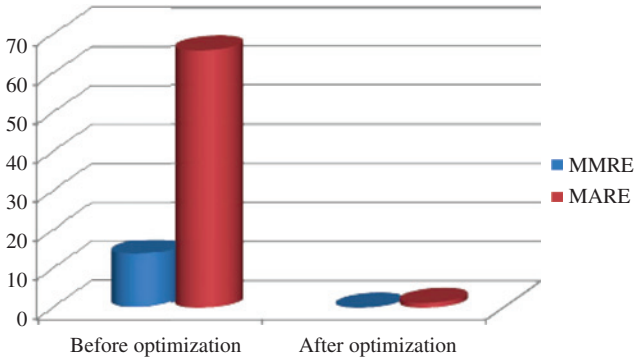

Figure 4: Graphical Representation of the MMRE and MARE Values Before and After Optimization for the Desharnais Dataset.

Table 4: Analysis of the Estimated Effort for the Desharnais Dataset.

Original effort	Effort estimated by our proposed method	Effort estimated by neural network with ABC [13, 14]	Effort estimated by neural network with MCS	Effort estimated by neural network (without optimization)
5152	5134.96	5222.7	5205.704	5345.89
5635	5655.768	5727.423	5602.63	5876.12
805	804.26	895.45	781.991	700.34
3829	3834.1234	3881.1	3901.637	3705.05
2149	2151.434	2123.97	2247.38	2048.31

Table 5: Comparative Analysis of MMRE to Other Methods.

Methods	Desharnais	NASA 93	NASA 60
Proposed method	0.04101	0.0781	0.0562
Neural network with ABC [13, 14]	0.6428	0.695	0. 515
Neural network with MCS	0.3145	0.79	0.734
Fuzzy method	0.32651	0.478	0.546
GA-based method	0.0947	0.125	0.173

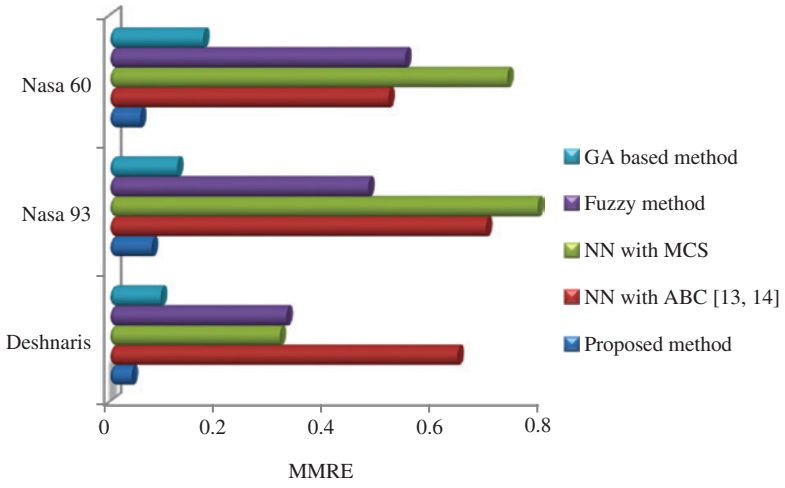

Figure 5: Graphical Representation of Comparison of the MMRE Value for the Proposed and Existing Methods.

This will optimally estimate the effort with minimum MMRE. The values are tabulated in Table 5, where the MMRE is given in percentage.

Figure 5 shows the graphical representation of the comparative results between our proposed method and the existing method based on MRE and MMRE measurements. Here, the comparison in terms of the MMRE is made between our proposed method, in our previous work where ABC is used, and the existing methods [13, 14]. The graphical representation demonstrates that our proposed method outperforms the existing methods in terms of efficiency.

5 Conclusion

In this proposed work, we use MFCM for clustering the dataset. Once the clustering is done, various rules are obtained and these rules are given as the input to the neural network. Here, we modify the neural network by incorporating optimization algorithms. The optimization algorithms employed here are the ABC, MCS, and hybrid ABC-MCS algorithms. Hence, we obtain three optimized sets of rules that are used for the effort estimation process. The performance of our proposed method is investigated using parameters such as the MARE and MMRE. The experimental outcomes have demonstrated that our proposed method outperforms the existing method in estimating the software effort more precisely.

In the future, we can examine the software effort based on the neuroticism characters of employees. The neuroticism characters highly affected the effort and cost of the software. Neuroticism characters are important to model because they have been shown to impact a range of cognitive, perceptual, and behavioral processes, such as memory recall (mood-congruent recall), learning, psychological disorders (depression), and decision making. The characters such as anger, anxiety, skill, and joy can be used to estimate the effort.

Bibliography

[1] R. Abdukalykov, I. Hussain, M. Kassab and O. Ormandjieva, Quantifying the impact of different non-functional requirements and problem domains on software effort estimation, in *Proc. 9th International Conference on Software Engineering Research, Management and Applications*, 2011.

[2] I. Attarzadeh, A. Mehranzadeh and A. Barati, Proposing an enhanced artificial neural network prediction model to improve the accuracy in software effort estimation, in *Proc. 4th International Conference on Computational Intelligence, Communication Systems and Networks*, 2012.

[3] M. Azzeh and A. B. Nassif, Analogy-based effort estimation: a new method to discover set of analogies from dataset characteristics, *IET Softw.* **9** (2015), 39–50.

[4] V. K. Bardsiri, D. N. A. Jawawi, S. Z. M. Hashim and E. Khatibi, Increasing the accuracy of software development effort estimation using projects clustering, *IET Softw.* **6** (2012), 461–473.

[5] Z. Dan, Improving the accuracy in software effort estimation using artificial neural network model based on particle swarm optimization, in *Proc. IEEE International Conference on Service Operations and Logistics, and Informatics (SOLI)*, 2013.

[6] K. Dejaeger, W. Verbeke, D. Martens and B. Baesens, Data mining techniques for software effort estimation: a comparative study, *IEEE Trans. Softw. Eng.* **38** (2012), 375–397.

[7] M. El Bajta, Analogy-based software development effort estimation in global software development, in *Proc. IEEE 10th International Conference on Global Software Engineering Workshops*, 2015.

[8] K. Hamdan, H. El Khatib, J. Moses and P. Smith, A software cost ontology system for assisting estimation of software project effort for use with case-based reasoning, *Innov. Inf. Technol.* (2006).

[9] D. Karaboga and B. Akay, A comparative study of artificial bee colony algorithm, *J. Appl. Math. Comput.* **214** (2009), 108–132.

[10] D. Karaboga and C. Ozturk, Fuzzy clustering with artificial bee colony algorithm, *J. Sci. Res. Essays* **5** (2010), 1899–1902.

[11] E. Kocaguneli, T. Menzies, J. Keung, D. Cok and R. Madachy, Active learning and effort estimation: finding the essential content of software effort estimation data, *IEEE Trans. Softw. Eng.* **39** (2013), 1040–1053.

[12] W.-T. Lee, J. Lee, K.-H. Hsu and J. Y. Kuo, Applying software effort estimation model based on work breakdown structure, in *Proc. 6th International Conference on Genetic and Evolutionary Computing*, 2012.

[13] S. Malathi and S. Sridhar, Estimation of effort in software cost analysis for heterogenous dataset using fuzzy analogy, *Int. J. Comput. Sci. Inf. Security* **10** (2012).

[14] A. L. I. Oliveira, P. L. Braga, R. M. F. Lima and M. L. Cornelio, GA-based method for feature selection and parameters optimization for machine learning regression applied to software effort estimation, *Inf. Softw. Technol.* **52** (2010), 1155–1166.
[15] J. Popovic, D. Bojic and N. Korolija, Analysis of task effort estimation accuracy based on use case point size, *IET Softw.* **9** (2015), 1–8.
[16] S. M. Satapathy, M. Kumar and S. K. Rath, Class point approach for software effort estimation using soft computing techniques, *ACM SIGSOFT Softw. Eng.* **39** (2014), 1–6.
[17] S. M. Satapathy, B. P. Acharya and S. K. Rath, Early stage software effort estimation using random forest technique based on use case points, *IET Softw.* **10** (2016), 10–17.
[18] Y.-S. Seo, K.-A. Yoon and D.-H. Bae, Improving the accuracy of software effort estimation based on multiple least square regression models by estimation error-based data partitioning, in *Proc. 16th Asia-Pacific Software Engineering Conference*, 2009.
[19] T. Wijayasiriwardhane, R. Lai and K. C. Kang, Effort estimation of component-based software development – a survey, *IET Softw.* **5** (2011), 216–228.