# Software Cost Estimation using Computational Intelligence Techniques

J.S.Pahariya
jankisharanpahariya@yahoo.com

V. Ravi*
rav_padma@yahoo.com

M. Carr
mahil.carr@gmail.com

*Institute for Development and Research in Banking Technology, Castle Hills Road #1, Masab Tank, Hyderabad 500 057, AP, India*

## ABSTRACT

*This paper presents computational intelligence techniques for software cost estimation. We proposed a new recurrent architecture for Genetic Programming (GP) in the process. Three linear ensembles based on (i) arithmetic mean (ii) geometric mean and (iii) harmonic mean are implemented. We also performed GP based feature selection. The efficacy of these techniques viz Multiple Linear Regression, Polynomial Regression, Support Vector Regression, Classification and Regression Tree, Multivariate Adaptive Regression Splines, Multilayer FeedForward Neural Network, Radial Basis Function Neural Network, Counter Propagation Neural Network, Dynamic Evolving Neuro–Fuzzy Inference System, Tree Net, Group Method of Data Handling and Genetic Programming has been tested on the International Software Benchmarking Standards Group (ISBSG) release 10 dataset. Ten-fold cross validation is performed throughout the study. The results obtained from our experiments indicate that new recurrent architecture for Genetic Programming outperformed all the other techniques.*

## Keywords

Multiple Linear Regression (MLR), Polynomial Regression, Support Vector Regression (SVR), Classification and Regression Tree (CART), Multivariate Adaptive Regression Splines (MARS), Multilayer FeedForward Neural Network (MPFF), Radial Basis Function Neural Network (RBF), Counter Propagation Neural Network (CPNN), Dynamic Evolving Neuro–Fuzzy Inference System (DENFIS), Tree Net, Group Method of Data Handling (GMDH) and Genetic Programming (GP).

* Corresponding author. Phone: +91 40 23534981 Ext. 2042; FAX: +91 40 23535157.

## 1. Introduction

The estimation of software development cost is one of the most critical problems in software engineering. Software cost development is related to how long and how many people are required to complete a software project. Software development has become an essential question [1] because many projects are still not completed on schedule, with under or over estimation of efforts leading to their own particular problems [2]. Therefore, in order to manage budget and schedule of software projects [3], various software cost estimation models have been developed. A major problem of the software cost estimation is first obtaining an accurate size estimate of the software to be developed [4] because size is the most important single cost driver [5]. Thus, an important objective of the software engineering community has been to develop useful models that can explain the software development life cycle and accurately estimate the cost of software development [6, 7]. Development cost tends to increase with project complexity and hence accurate cost estimates are highly desired during the early stages of development [8].

The main objective of the present work is to propose a new computational intelligence model that estimates the software cost accurately. The rest of the paper is organized as follows. Section 2 reviews the research done in the field of software cost estimation. Section 3 overviews the data description and data preparation Section 4 overviews the techniques applied in this paper. Section 5 overview the recurrent architecture based on GP developed in this paper. Section 6 presents the results and discussions. Finally, Section 7 concludes the paper.

## 2. Literature review

Various effort estimation models have been developed over the last four decades. The most commonly used methods for predicting software development efforts are Function Point Analysis and COnstructive COst MOdel (COCOMO)[4]. Function Point Analysis was developed first by Albrecht (1979) (www.IFPUG.Org). Function point analysis is a method of quantifying the size and complexity of a software system in terms of the functions that the system delivers to the user [9]. The function does not depend on the programming languages or tools used to develop a software project [1]. COCOMO is developed by the Boehm [2]. It is based on linear-least-squares regression. Using line of code (LOC) as the unit of measure for software size itself contains so many problems [10]. These methods failed to deal with the implicit non-linearity and interactions between the characteristics of the project and effort [11, 12].

In recent years, a number of alternative modeling techniques have been proposed. They include artificial neural networks, analogy-based reasoning, and fuzzy system and ensemble techniques. Ensemble is used to combine the result of individual methods [13, 14]. In analogy-based cost estimation, similarity measures between a pair of projects play a critical role [15]. This type of model calculates distance between the software project being estimated and each of the historical software projects and then retrieves the most similar project for generating an effort estimate [16]. Further, Lefley and Shepperd [21] applied genetic programming to improve software cost estimation on public datasets with great success. Later, Vinaykumar et al. [6] used wavelet neural networks for the prediction of software cost estimation. Unfortunately the accuracy of these models is not satisfactory so there is always a scope for new software cost estimation techniques.

## 3. Data Description Data Preparation

The ISBSG is administered from Australia (http://www.isbsg.org). Entire ISBSG-10 dataset contains information about 4109 projects. The dataset consist of 18 attributes. These attributes also divided into sub-attributes thus, number of attributes is 105. In this paper, we are predicting the summary of work effort, i.e. manpower required to complete the work. Once we know the effort, we can easily calculate the time and the cost of the software. However, before applying these statistical and intelligence techniques to the dataset, there are a number of issues to be taken into consideration during data cleaning and data preparation.

The first cleaning step was to remove the projects having null values for the attribute named Summary of Work Effort. Secondly regarding summary of work effort only 1538 project values are given for the five attributes viz Input count, Output, Enquiry, File and Interface. If we consider more attributes then we get only a few projects which are not sufficient for machine learning techniques. Hence we considered 1538 projects values with five attributes to do train and test several intelligent models. Finally, we normalized the data set.

## 4. Overview of the techniques employed

The following techniques are applied to predict summary of work effort:
Group Method of Data Handling (GMDH), Genetic Programming (GP), Counter Propagation Neural Network (CPNN), Multiple Linear Regression (MLR), Polynomial Regression, Support Vector Regression (SVR), Classification and Regression Tree (CART), Multivariate Adaptive Regression Splines (MARS), Multilayer FeedForward Neural Network (MPFF), Radial Basis Function Neural Network (RBF), Dynamic Evolving Neuro–Fuzzy Inference System (DENFIS), and Tree Net.

### 4.1 Group Method of Data Handling (GMDH)

Ivakhnenko [22] invented GMDH as an inductive learning algorithm for complex system modeling. GMDH is a self-organizing approach based on sorting-out of gradually complicated models and evaluation of them by external criterion on separate part of data sample [17]. GMDH model with multiple inputs and one output is a subset of components of the base function

$$Y(x_1, x_2, \ldots., x_n) = a_0 + \sum_{i=1}^{m} a_i f_i$$

Where $f$ is an elementary function depend on different sets of inputs, $a_i$ represents coefficients and $m$ represent number of base function components. In order to find the best solution GMDH algorithm considers various component subsets of the base function called Partial Models. Coefficients of these models are estimated by the least squares method.

### 4.2 Genetic Programming (GP)

Genetic programming (GP) is an extension of Genetic Algorithms (GA). GP is a search methodology belonging to the family of Evolutionary Computation (EC). Genetic programming starts from a high-level statement of 'what needs to be done' and automatically creates computer programs to solve the problem. This population of programs is progressively evolved over a series of generations [18]. GP randomly generates an initial population of solutions. The initial population is manipulated using various genetic operators to produce new populations. These operators include reproduction, crossover, mutation, dropping condition, etc. A high-level description of GP algorithm can be divided into a number of sequential steps:

- Create a random population of programs, or rules, using the symbolic expressions provided as the initial population.
- Evaluate each program or rule by assigning a fitness value according to a predefined fitness function.
- Use reproduction operator to copy existing programs into the new generation.
- Generate the new population with crossover, mutation, or other operators from a randomly chosen set of parents.
- Repeat steps 2 onwards for the new population until a predefined termination criterion has been satisfied, or a fixed number of generations have been completed.
- The solution to the problem is the genetic program with the best fitness within all the generations.

We used the GP implementation available at http://www.rmltech.com.

## 4.3 Counter Propagation Neural Network (CPNN)

The counter propagation network is a competitive network and given by Hecht-Nielsen [23]. It has a two-layered architecture. One layer is the competitive viz Kohonen layer and the other is the Grossberg layer which acts as the output layer. The input layer in CPNN performs the mapping of the multidimensional input data into lower dimensional array. The mapping is performed by use of competitive learning, which employs winner-takes-it-all strategy [19]. The training process of the CPNN is partly similar to that of Kohonen self-organizing maps. The Grossberg layer performs supervised learning.

## 4.4 Support Vector Regression (SVR)

The SVR is a powerful learning algorithm based on recent advances in statistical learning theory proposed by Vapnik [24]. SVR are learning systems that use a hypothesis space of linear functions in a high-dimensional space, trained with a learning algorithm from optimization theory that implements a learning bias derived from statistical learning theory. SVR uses a linear model to implement non-linear class boundaries by mapping input vectors non-linearly into a high dimensional feature space using kernels. The training examples that are closest to the maximum margin hyperplane are called support vectors. All other training examples are irrelevant for defining the binary class boundaries. The support vectors are then used to construct an optimal linear separating hyperplane (in case of pattern recognition) or a linear regression function (in case of regression) in this feature space. The support vectors are conventionally determined by solving a quadratic programming (QP) problem. We used the SVR implementation available at http://rapid-i.com/content/view/26/84/.

## 4.5 Classification and Regression Tree (CART)

CART was introduced by Breiman et al. [25] can solve both classification and regression problems (http://salford-systems.com). Decision tree algorithms induce a binary tree on a given training data, resulting in a set of ‘if–then’ rules. These rules can be used to solve the classification or regression problem. The key elements of a CART analysis [25] are a set of rules for: (i) splitting each node in a tree, (ii) deciding when a tree is complete; and (iii) assigning each terminal node to a class outcome (or predicted value for regression). We used the CART implementation available at http://salford-systems.com.

## 4.6 Multivariate Adaptive Regression Splines (MARS)

Multivariate adaptive regression splines (MARS) introduced by Friedman [26]. MARS is an innovative and flexible modeling tool that automates the building of accurate predictive models for continuous and binary dependent variables. It excels at finding optimal variable transformations and interactions, the complex data structure that often hides high-dimensional data. This approach to regression modeling effectively uncovers important data patterns and relationships that are difficult, if not impossible, for other methods to reveal. We used the MARS implementation available at http://salford-systems.com.

## 4.7 Dynamic Evolving Neuro–Fuzzy Inference System (DENFIS)

DENFIS was introduced by Kasabov and Song [27]. DENFIS evolve through incremental, hybrid (supervised/unsupervised) learning, and accommodate new input data, including new features, new classes, etc., through local element tuning. New fuzzy rules are created and updated during the operation of the system. At each level, the output of DENFIS is calculated through a fuzzy inference system based on most activated fuzzy rules, which are dynamically chosen from a fuzzy rule set. A set of fuzzy rules can be inserted into DENFIS before or during its learning process. Fuzzy rules can also be extracted during or after the learning process. Student version of the NewCom tool obtains at http://www.aut.ac.nz/research/research_institutes/kedri/research_centres/centre_for_data_mining_and_decision_support_systems/neucom.htm#download was used in this paper for DENFIS and MLR.

### 4.8 Tree Net

Tree Net was introduced by Friedman [28]. It makes use of a new concept of 'ultra slow learning' in which layers of information are gradually peeled off to reveal structure in data. TreeNet models are typically composed of hundreds of small trees, each of which contributes just a tiny adjustment to the overall model. TreeNet is insensitive to data errors and needs no time-consuming data preprocessing or imputation of missing values. TreeNet is resistant to overtraining and is faster than a neural net. TreeNet available at http://salford-systems.com was used.

### 4.9 Radial Basis Function Neural Network (RBF)

RBF was introduced by Moody and Darken [29], has both unsupervised and supervised phases intandem. In the unsupervised phase, input data will be clustered and cluster details are subsequently sent to hidden neurons, where radial basis functions of the inputs are computed by making use of the center and the standard deviation of the clusters. Gaussian radial basis functions are the most commonly used functions. The learning between hidden layer and output layer is of supervised learning type where ordinary least squares technique is used. As a consequence, the weights of the connections between the kernel layer and the output layer are determined. KNIME tool available at http://www.knime.org was used for MLP, RBF, and polynomial regression.

## 5. Proposed Method.

The fundamental assumption in computation intelligence paradigm is that hybrid intelligent techniques tend to outperform the stand-alone techniques. We proposed new recurrent architecture for Genetic Programming (RGP) in which output of the GP is feedback as an input to the GP. This is analogous to recurrent networks having feedback loop where output can be fed back as input [20].

The flow diagram of the recurrent architecture for Genetic Programming (RGP) is depicted in Figure 1.It is observed that there are some features in the dataset that are contributing negatively to the prediction accuracy of all the predictors. Hence we resolved to feature selection. We used GP for feature selection. Using GP based feature selection we selected four most important variables for training. We proposed two hybrids based on feature selection

(i) GP-GP
(ii) GP-RGP

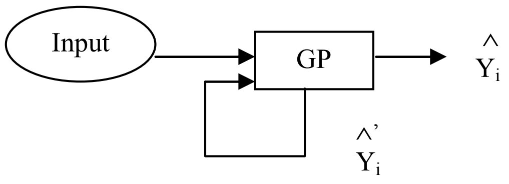

**Figure.1 Recurrent architecture for Genetic Programming (RGP)**

First technique is used for feature selection and second technique is used for prediction.

GP-GP hybrid depicted in Figure 2 and GP-RGP hybrid depicted in Figure 3.

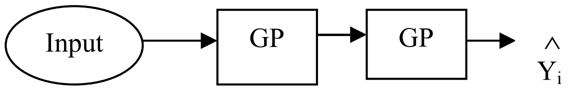

**Figure.2 GP-GP hybrid**

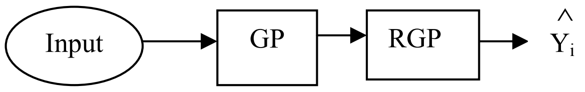

**Figure.3 GP-RGP hybrid**

We also implemented linear ensemble systems. Ensemble systems exploit the unique strengths of each constituent model and combine them in same way. For constructing ensemble system we have chosen three best techniques from stand-alone techniques that are GMDH, GP, and CPNN. We constructed ensembles using three methods. Arithmetic Mean (AM) Harmonic Mean (HM) Geometric Mean (GM). Ensemble system is depicted in Figure 4.

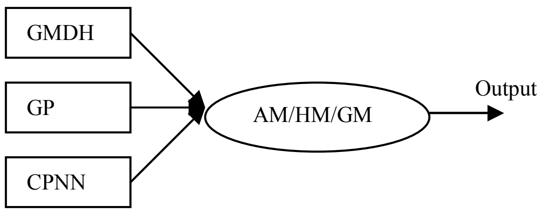

**Figure.4 Ensemble System**

## 6. Results and discussion

We used ISBSG data set, which contains 1538 projects having five independent variables and one dependent variable. We compared the performance of cost estimation models on the basis of root mean square error (RMSE), which defined as follows:

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(Effort_{actual_i} - Effort_{estimated_i}\right)^2}$$

Where n is the number of projects.

We observed that GP-GP and GP-RGP have outperformed all other stand-alone techniques. In between then GP-GP and GP-RGP, GP-RGP is found to be better. The results are presented in Table 1.

**Table 1: Average RMSE of 10 fold cross validation**

| SN | METHOD | RMSE (TEST) |
|---|---|---|
| 1 | GP-RGP | **0.03345** |
| 2 | GP-GP | 0.03676 |
| 3 | GMDH | 0.03784 |
| 4 | GP | 0.03794 |
| 5 | CPNN | 0.04499 |
| 6 | CART | 0.04561 |
| 7 | TREENET | 0.04565 |
| 8 | MLP | 0.04817 |
| 9 | MLR | 0.04833 |
| 10 | DENFIS | 0.04837 |
| 11 | MARS | 0.0487 |
| 12 | SVR | 0.0492 |
| 13 | RBF | 0.05167 |
| 14 | POLYNOMIAL REGRESSION | 0.05327 |

We notice that AM based ensemble system has outperformed GM based and HM based ensemble techniques. The results are presented in Table 2.

**Table 2: Average RMSE of ensemble models**

| SN | METHOD | RMSE (TEST) |
|---|---|---|
| 1 | AM | **0.0421** |
| 2 | GM | 0.04403 |
| 3 | HM | 0.0455 |

Our proposed recurrent architecture for GP (RGP) model yielded least RMSE value of 0.03275. We observed that RGP has outperformed all other hybrid techniques. The results are presented in Table 3.

**Table 3: Average RMSE of hybrids models**

| SN | METHOD | RMSE (TEST) |
|---|---|---|
| 1 | RGP | **0.03275** |
| 2 | GP-RGP | 0.03345 |
| 3 | GP-GP | 0.03676 |

Further, we performed t-test between the average RMSEs over all the 10 folds obtained by the top three methods viz., RGP, GP-RGP and GP-GP. The t-statistic value between RGP and GP-RGP is 0.866 whereas between RGP and GP-GP is 1.78. We can observe that t-statistic value between RGP and GP-GP is greater than 1.73, which is the tabulated t-test value at $n_1+n_2-2=10+10-2=18$ degrees of freedom at 10% level of significance. Also, and the t-statistic value between RGP and GP-RGP is less than 1.73. Therefore, we conclude that the superior performance obtained by RGP vis-à-vis GP-GP is statistically significant. Also we note that the difference in performance between RGP and GP-RGP is not statistically significant.

## 7. Conclusions

This paper presents recurrent architecture for on Genetic Programming (RGP), for software cost estimation. Throughout the study ten-fold cross validation is performed. We tested a host of techniques on the ISBSG dataset. The proposed RGP model out performed all other techniques. We notice that GP-RGP stood second with an average RMSE value of 0.03345. Hence we conclude that the RGP model is the relatively best predictor among all other techniques.

## REFERENCES

[1] A.J. Albrecht and J.E. Gaffney, "Software function, source lines of code, and development effort prediction: a software science validation," IEEE Transactions on Software Engineering, 1983, pp. 639–647.

[2] B.W.Boehm, "Software Engineering Economics," Prentice-Hall, Englewood Cliffs, NJ, USA, 1981.

[3] L.H.Putnam, "A general empirical solution to the macro software sizing and estimation problem," IEEE Transactions on Software Engineering, July 1978, pp. 345–361.

[4] B. Kitchenham, L.M. Pickard, S. Linkman and P.W. Jones, "Modeling software bidding risks," IEEE Transactions on Software Engineering, 2003, pp. 542–554.

[5] J. Verner and G. Tate, "A Software Size Model," IEEE Transactions on Software Engineering, 1992, No. 4.

[6] K. Vinaykumar, V. Ravi, M. Carr and N. Rajkiran, "Software cost estimation using wavelet neural networks," Journal of Systems and Software, 2008, pp. 1853-1867

[7] T. Foss, E. Stensrud, B. Kitchenham and I. Myrtveit, "A simulation study of the model evaluation criterion MMRE," IEEE Transactions on Software Engineering, 2003, pp. 985–995.

[8] Z. Xu and T.M. Khoshgoftaar "Identification of fuzzy models of cost estimation," Fuzzy Sets and Systems, 2004, pp. 141-163

[9] J.E. Matson, B.E Barrett and J.M. Mellichamp, "Software development cost estimation using function points," IEEE Transactions on Software Engineering, 1994, pp. 275–287.

[10] A.Idri, T.M. Khosgoftaar and A. Abran, "Can neural networks be easily interpreted in software cost estimation," World Congress on Computational Intelligence, Honolulu, Hawaii, USA, 2002, pp. 12–17.

[11] A.R. Gray, "A simulation-based comparison of empirical modeling techniques for software metric models of development effort," In: Proceedings of ICONIP, Sixth International Conference on Neural Information Processing, Perth, WA, Australia, 1999, pp. 526–531.

[12] X.Huang, L.F.Capetz,J. Ren and D.Ho, "A neuro-fuzzy model for software cost estimation," Proceedings of the third International Conference on Quality Software, 2003, pp. 126-133 .

[13] K.K. Aggarwal, Y. Singh, P.Chandra and M.Puri, "An expert committee model to estimate line of code," ACM New York, NY, USA, 2005, pp. 1-4.

[14] K. Vinay Kumar, V. Ravi and Mahil Carr, "Software Cost Estimation using Soft Computing Approaches," Handbook on Machine Learning Applications and Trends: Algorithms, Methods and Techniques, Eds. E. Soria, J.D. Martin, R. Magdalena, M.Martinez, A.J. Serrano, IGI Global, USA, 2009.

[15] Y.F. Li, M. Xie and T.N. Goh, "A study of project selection and feature weighting for analogy based software cost estimation," Journal of Systems and Software, 2009, pp. 241–252

[16] N.H. Chiu and S.J.Huang, "The adjusted analogy-based software effort estimation based on similarity distances," System and Software, 2007, pp.628-640

[17] D. Srinivasan, "Energy demand prediction using GMDH networks," Neurocomputing, 2008, pp. 625-629.

[18] R. Poli, W.B. Langdon and J.R. Koza, "A field guide to Genetic Programming," publisher- Lulu.com, United Kingdom, 2008.

[19] I. Kuzmanovski and M. Novic, "Counter-Propagation Neural Networks in Matlab," Chemometrics and Intelligent Laboratory Systems, 2008, pp. 84-91

[20] G. Dematos, M. S Boyd, B. Kermanshahi, N. Kohzadi and I. Kaastra, "Feedforward versus recurrent neural networks for forecasting monthly japanese yen exchange rates," Asia-Pacific Financial Markets, 1996, pp. 59-75.

[21] M. Lefley and M. J. Shepperd, "Using Genetic Programming to Improve Software Effort Estimation Based on General Data Sets", LNCS, Genetic and Evolutionary Computation — GECCO 2003, ISBN: 978-3-540-40603-7, page-208.

[22] A.G. Ivakhnenko, "The group method of data handling -- a rival of the method of stochastic approximation", Soviet Automatic Control, 13(3), 1966, pp. 43-55.

[23] R. Hecht-Nielsen, Counterpropagation Networks, Applied Optics*,* v.26, #23, 1987, pp. 4979-4984.

[24] V.N. Vapnik, 'Statistical Learning Theory", John Wiley, New York, 1998.

[25] L. Breiman, J. H. Friedman, R. A. Olshen, C. J. Stone, "Classification and Regression Trees", Wadsworth, Belmont-CA, 1984.

[26] J.H. Friedman, "Multivariate adaptive regression splines", Ann Stat*,* 1991, 19, 150.

[27] N.K. Kasabov, Q. Song, "DENFIS: dynamic evolving neural-fuzzy inference system and its application for time-series prediction". IEEE Transactions on Fuzzy Systems 10 (2), 2002, pp. 144–154.

[28] J.H. Friedman, "Stochastic gradient boosting", Stanford, Statistics Department, Stanford University, 1999.

[29] J. Moody, C.J. Darken, "Fast learning in networks of locally tuned processing units', Neural Computation. 1, 1989, pp. 281–294.