We are IntechOpen, the world's leading publisher of Open Access books Built by scientists, for scientists

4,800
Open access books available

122,000
International authors and editors

135M
Downloads

Our authors are among the

154
Countries delivered to

TOP 1%
most cited scientists

12.2%
Contributors from top 500 universities

CLARIVATE ANALYTICS BOOK CITATION INDEX INDEXED

WEB OF SCIENCE™

Selection of our books indexed in the Book Citation Index in Web of Science™ Core Collection (BKCI)

Interested in publishing with us?
Contact book.department@intechopen.com

Numbers displayed above are based on latest data collected.
For more information visit www.intechopen.com

9

Size-Based Software Cost Modelling with Artificial Neural Networks and Genetic Algorithms

Efi Papatheocharous[1] and Andreas S. Andreou[2]
[1]*Department of Computer Science, University of Cyprus*
[2]*Department of Electrical Engineering and Information Technology, Cyprus University of Technology*
Cyprus

1. Introduction

Accurate software cost estimation has always been a major concern especially for people involved in project management, resource control and schedule planning. A high-quality and reliable development effort estimate could provide more efficient planning and control over the whole software process and guide a project to success. As literature shows many researchers proposed a plethora of methods and techniques to model the relationship between software size and the actual development costs (Jørgensen & Shepperd, 2007). However, the track record of IT projects shows that often a large number fails. Most IT experts agree that such failures occur more regularly than they should (Charette, 2005). According to the 10th edition of the annual CHAOS report from the Standish Group that studied over 40,000 projects in 10 years, it seems that success rates increased to 34% and failures declined to 15% of the projects. Even though success rates increased, still, 51% of the projects overrun time/budget, lack critical features and requirements and/or important quality requirements are compromised. Furthermore, average software costs are apparently overrun by 43% (Software Magazine, 2004). One of the main reasons for these figures is failure to estimate the actual effort required to develop a software project.

A reliable software cost estimation model has always been a major challenge and demand for project managers at the initiation phase of the project and also an important asset for the whole process of software development. In addition, there is a large discussion on the discovery of the relationship between cost drivers and effort, especially of one of the most critical cost factors, namely software size (Sommerville, 2007). The aforementioned modelling and estimation problem is further amplified due to the high level of complexity and uniqueness of each project. Estimating software costs, as well as deciding on assessing the appropriate cost drivers, remain difficult issues that need to be addressed. Such issues are constantly at the forefront of the project management's interests from the initiation of the project until the system is delivered. Cost estimates, even for well-planned projects, are hard to make and will probably concern project managers long before the problem is adequately solved.

Over the years software cost estimation has attracted considerable research attention and many techniques have been developed to effectively predict software costs (Briand &

Wieczorek, 2002; Moløkken & Jørgensen, 2003; Jørgensen & Shepperd, 2007). The dominant techniques during the past decades involved Regression since it was applied in well-known software cost estimation models (such as the COnstructive COst MOdel, COCOMO) to capture the relationship between cost drivers and effort. During the last years, analogy-based, expert judgement, decision trees, neural networks, probabilistic and other approaches arose in software cost estimation studies (Jørgensen & Shepperd, 2007). Nevertheless, no single solution has yet been proposed to adequately address the problem. Typically, the amount and complexity of the development effort proportionally drives software costs. However, as other factors, such as technology, team and manager skills, software quality, project size, also affect the development process it becomes even more difficult to assess the actual costs.

A commonly investigated approach is to estimate as accurately as possible some of the fundamental characteristics related to cost, such as effort, usually measured in person-months, through past empirical project examples. In addition, it is also preferable to measure a condensed set of several attributes which can be more informative (descriptive) in regards to effort and then use them to estimate the actual effort. However, previous research identified the lack of standard definitions in software terminology and the presence of inconsistencies in empirical data samples, where it was also concluded that models with too many variables and parameters are very hard to calibrate (Miyazaki et al., 1994). For this reason, building models that focus only on a small set of significant attributes is more practical.

Software size is commonly recognised as one of the most important factors affecting the amount of effort required to complete a project (Fenton & Pfleeger, 1997). Software size in terms of actual code length is considered a fairly subjective metric as it depends on the development language and the code generated by tools or re-used in software development. Also, software size is impractical in providing early effort estimates, that is, at the beginning of a project, mainly because it is unknown until the project's source code is actually written. Therefore, after software specification is finalised, the estimation of software size based on the outline of the project is a fundamental activity. Also, it is very critical to carry out estimation of software size after the initial phases of development and the success of the final effort approximation may probably depend greatly on its value. This chapter looks more closely into the relations of these two most significant parts of software cost estimation.

More specifically, some researchers have investigated various cost models using size to estimate effort (e.g., Wittig & Finnie, 1997; Dolado, 2001) whereas others have directed their efforts towards defining concise methods and measures to estimate software size from the early project phases (e.g., Park, 2005; Albrecht, 1979). The present work is more relevant to the former, aspiring to provide size and effort-based estimations and modelling the relationship between size, as this is expressed in terms of Lines of Code (LOC) or Function Points (FP), and development effort. The size of the programs under development is considered known for a collection of past, historical projects and in some cases their respective effort value is used to train the Artificial Neural Networks (ANN) models proposed. A set of projects is intentionally left out from the training process and is used to verify the generalisation of the models. Therefore, the main target of this investigation is to use the size values of new projects and utilise the robust cost models developed to approximate their effort value. The proposed models also make use of the effort values for the projects employed in the training of the models to investigate accuracy performance. The hypothesis is that once a robust relationship between size and effort is established by means of a model, then it can be used along with the size estimations to predict effort of new

projects more accurately. In addition, a Genetic Algorithm (GA) is constructed to calibrate the model's architecture.

Thus, in this work we study the potentials of developing a software cost model using computational intelligence techniques relying only on size and effort data. The core of the model proposed consists of Artificial Neural Networks (ANN). Our principal investigation is to determine which size-related metric between LOC and FP is more descriptive of effort with the aid of ANN. The initial approach builds ANN using size-related input data and the architecture is empirically defined. The second approach investigates a more complicated issue, i.e., which combination of size and/or effort related data of historical projects and which ANN architecture provides better effort approximations for the new projects. Essentially what is investigated in this approach is the size of a sliding-window used to extract inputs and the ANN topology. The experiments conducted suggest that quite promising accuracy results can be obtained and that if we specify the appropriate ANN architecture for each dataset, even more improved effort approximations may be achieved. Therefore, in the third approach the sliding-window length and the ANN architecture are optimised with the use of a Genetic Algorithm (GA). The GA evolves the ANN structure with the appropriate number and type of size-related inputs (i.e., LOC or FP), as well as the optimal internal hidden neuron architecture, to predict effort as accurately as possible. The inputs used to train and test the ANN are in this case: project size measurements (either Lines of Code (LOC) or Function Points (FP)), and/or the associated effort to predict the subsequent in series, unknown project effort. In addition, a Regression cost estimation model is presented as a benchmark to assess the performance of the model materialising estimations of the dependent variable (effort) using the same aforementioned training and testing samples, so that the proposed models are compared to a classic method.

The rest of the paper is organised as follows: Section 2 presents a literature overview of relative research on size-based software cost estimation and especially focuses on machine learning techniques utilising ANN. Section 3 provides a description of the datasets and performance metrics used in the experiments following in Section 4. Section 4 includes the application of an ANN cost estimation model and describes an investigation of further improvements of the model proposing a hybrid algorithm to evaluate the optimal input methods and architectures for the datasets. In addition, this section summarises the results of each respective approach proposed and presents a comparison of the results to a classic Regression. Section 5, concludes with the overall remarks and findings of this work, discusses a few limitations and suggests future research steps.

2. Related work

Several techniques have been investigated for software cost estimation, especially data-driven artificial intelligence techniques, such as neural networks, evolutionary computing, regression trees, rule-based induction as they present several advantages over other, traditional approaches like regression. Most of the relevant studies investigate, among other issues, the identification and realisation of the most important factors that influence software costs. This section focuses on related work mainly of size-based, neural network models proposed for software cost estimation.

To begin with, most size-based models consider either the number of lines written for a project (called Lines of Code (LOC) or thousands of Lines of Code (KLOC)) used in models such as the COCOMO (Boehm, 1981; Boehm et al., 1997), or the number of Function Points

(FP) used in models such as Albrecht's Function Point Analysis (FPA) (Albrecht & Gaffney, 1983). In size-based software cost estimation models essentially, the size of LOC or FP is estimated at the early stages of development. The LOC metric has been primarily used in models such as the COCOMO (Boehm, 1981; Boehm, 1997) and SLIM (Putnam, 1978; Putnam & Myers, 1992; Putnam & Myers, 2003), models first proposed in the previous decades. LOC was considered quite popular because it represents a direct measurement of the length of the software under development. However, the main weakness of the LOC metric is that it cannot be estimated accurately from the beginning of the development phases of a project. Therefore, the FP metric was proposed in order to overcome this limitation (Albrecht, 1979) and estimate software size based on the requirements. FP basically represent a weighted sum of the following five factors: Input Count, Output Count, Logic Files Count, Inquiries Count and Interface Files Count. FP's advantage is that after requirements specification these factors become known and moreover different systems can be compared irrespectively of the technologies and languages used in development. However, there is a lot of discussion regarding the reliability and objectivity of both size metrics as different tools counting LOC depend on the language and definitions used and different practitioners counting FP for the same projects produce different results (Kemerer, 1993).

Many research studies investigate the potential of developing software cost prediction systems using different approaches, datasets and cost factors. Review articles, like the ones of Briand & Wieczorek (2002), Jørgensen & Shepperd (2007), include a detailed description of such studies. In this section we highlight some of the most important relevant studies dealing with size-based estimations.

Wittig and Finnie (1997) estimated effort using the backpropagation algorithm on ANN for the Desharnais and ASMA datasets, mainly using system size to determine it's relationship with effort. The approach yielded promising prediction results indicating, though, that the model required a more systematic development approach to establish the topology and parameter settings so as to obtain better results.

Dolado (2001) searched for the cost estimation equation of the relationship between size and effort by using Genetic Programming tree structures representing several classical equations, like the linear, power, quadratic, etc. The approach reached to moderately good levels of prediction accuracy results by using solely the size attribute and indicated that further improvements can be achieved.

Mittas et al. (2010) proposed the Demming Regression for modelling the relationship between software effort and size on the four datasets, Desharnais, COCOMO, Maxwell and Nasa93 based on the assumption that the observed values of the variables are measurements which coincide with the actual size values. Under this assumption the proposed technique estimated the regression coefficients and showed significant improvements in comparison to the classic Ordinary Least Squares regression.

In summary, the literature thus far, exhibits several research attempts focusing on measuring effort and size and accepting them as the key variables in cost estimation. Many studies indicate that ANN models are quite promising estimators or that they perform at least as well as other approaches. In the rest of this section we investigate such approaches.

Heiat (2002) compared the performance of two cost estimation techniques with respect to the type of language used for developing a range of projects. The finding of this work was that the ANN performed equally well with Regression for the sample projects that were implemented with a third generation programming language (3GL). However, experimenting with a less

homogeneous set of projects, that is, projects that were implemented with a third generation programming language (3GL) and others that were implemented with a fourth generation programming language (4GL), ANN outperformed Regression.

Idri et al. (2002) conducted two experiments using a backpropagation trained Multi-Layer Perceptron (MLP) ANN architecture on the COCOMO dataset, the outputs of which were mapped to a fuzzy rule-based system. Results indicated poor accuracy performance, while it is most likely that the experiments suffered from overfitting as an extreme number of iterations (300,000) were executed on just a small set of 63 samples.

Idri et al. (2004) investigated the use and interpretation of Radial Basis Function Networks (RBFN) in software cost estimation by mapping the ANN to a fuzzy rule-based system. Results on the COCOMO dataset indicated that the accuracy of the ANN depended heavily on the parameters of the middle layer and more specifically on the number of hidden neurons and the weight values.

Kumar et al. (2008) used Wavelet Neural Networks (WNN) for software development estimation and compared their effectiveness with MLP, RBFN, Multiple Linear Regression (MLR), Dynamic Evolving Neuro-Fuzzy Inference System (DENFIS) and Support Vector Machines (SVM) in terms of the *Mean Magnitude of Relative Error* (*MMRE)*. WNN seemed to outperform all other techniques.

Tronto et al. (2008) investigated the application of ANN and stepwise regression for software effort prediction. The experiments were conducted on the COCOMO dataset employing categorical variables whose impact was identified based on the work of Angelis et al. (2001) forming new categorical values. It was observed that there is a strong relationship between the success of a technique and the size of the learning dataset, the nature of the function for cost and other dataset characteristics (such as existence of outliers, collinearity and number of attributes).

Azzeh et al. (2010) investigated the impact of Grey Relational Analysis (GRA) integrated with Fuzzy set theory in a by-analogy estimation model and also compared it to ANN, CBR and MLR models using several public datasets, i.e., ISBSG, Desharnais, COCOMO, Albrecht and Kemerer. The Fuzzy GRA appeared to produce statistically more significant results than the rest of the models. Moreover, it effectively reduced the uncertainty of attribute measurement between two software projects and improved the way to handle both numerical and categorical data in similarity measurements.

Kaur et al. (2010) proved the effectiveness of ANN models for the NASA dataset compared to the Halstead, Walston-Felix, Bailey-Basili and Doty models, all of which are popular legacy models used in software cost estimation. Backpropagation ANN were used and reported as the most generalised networks currently in use that present good estimation capabilities.

Summarizing some of the findings of the relevant literature, we conclude that many researchers recognise the high prediction accuracy of ANN and their effectiveness in modelling the cost estimation environment. However, a deeper investigation on the topology and configurations of the ANN model, as well as the appropriate inputs required in each case, needs to be carried out, so that the complexity of the technique is not increased proportionally to the number of inputs and the complexity of the sample projects, and still accuracy is driven to better levels.

Subsequently, in this work we firstly aim to examine the potentials of ANN in software cost modelling and secondly to investigate the possibility of providing further improvements for such a model. Our goal is to inspect: (i) whether a suitable ANN model, in terms of input parameters, may be built; (ii) whether we can achieve sufficient estimates of software

development effort using only size or function based metrics on different datasets of empirical cost samples; (iii) whether a hybrid computational model, which consists of a combination of ANN and GA, may contribute to devising the ideal ANN architecture and set of inputs that meet some evaluation criteria. Our strategy is to exploit the benefits of computational intelligence in software cost modelling and provide a near to optimal effort predictor for impending new projects.

3. Datasets and performance metrics

A variety of historical software cost data samples coming from different datasets that are popular in software cost estimation empirical research were employed to provide a strong comparative basis with the results reported in other relevant studies. Also, the performance metrics used to assess the ANN's precision accuracy are described in this section.

3.1 Datasets description

The following datasets were selected to demonstrate and test the approach describing historical project data: COCOMO`81 (COC`81), Kemerer`87 (KEM`87), a combination of COCOMO`81 and Kemerer`87 (COKEM`87), Albrecht and Gaffney`83 (ALGAF`83) and finally Desharnais`89 (DESH`89).

The COC`81 (Boehm, 1981) dataset contains information about 63 software projects from different applications. Each project is described by the following 17 cost attributes: reliability, database size, complexity, required reusability, documentation, execution time constraint, main storage constraint, platform volatility, analyst capability, programmer capability, applications experience, platform experience, language & tool experience, personnel continuity, use of software tools, multi-site development and required schedule. Also, for the projects LOC is measured.

The second dataset, named KEM`87 (Kemerer, 1987) contains 15 software project records gathered by a single organisation in the USA, which constitute business applications written mainly in COBOL. The attributes of the dataset are: actual project's effort measured in man-months, duration, KLOC, unadjusted and adjusted FP's count. In addition, a combination of the two previous datasets was created, namely COKEM`87, to allow us to experiment with a larger but rather heterogeneous dataset.

The third dataset ALGAF`83 (Albrecht & Gaffney, 1983) contains information about 24 projects developed by the IBM DP service organisation. The datasets' characteristics correspond to the actual project effort, the KLOC, the number of inputs, the number of outputs, the number of master files, the number of inquiries and the FP's count.

The fourth dataset, DESH`89 (Desharnais, 1989), includes observations for more than 80 systems developed by a Canadian software development house at the end of 1980. The basic characteristics of the dataset account for the following: project name, development effort measured in hours, team's experience, project manager's experience, number of transactions processed, number of entities, unadjusted and adjusted FP, development environment and year of completion.

A major assumption of our work is that the measurements of some attributes provided for the projects in these datasets which are also used in our experiments, like for example the effort and the size-related factors of Lines of Code (LOC) and Function Points (FP), coincide with the actual values of developing the corresponding programs. However, since some of these software project metrics are conceptually subjective and lack standard definitions,

they depend on the person counting and the tools used to perform measurements, and thus, this high degree of subjectivity clearly makes the measurement of the software attributes and validation of the prediction systems for the attributes and effort problematic.

3.2 Performance metrics

The performance of the models was evaluated using a combination of common error metrics, namely the *Mean Relative Error* (*MRE*), the *Correlation Coefficient* (*CC*) and the *Normalized Root Mean Squared Error* (*NRMSE*), together with the Prediction at Level (*pred(l)*) and a devised Sign prediction (*Sign*) metric. These error metrics were employed to validate the model's forecasting ability by considering the difference between the actual and the predicted cost samples and their ascendant or descendant progression in relation to the actual values.

The *MRE*, given in equation (1), shows the prediction error focusing on the sample being predicted. $x_{act}(i)$ is the actual effort and $x_{pred}(i)$ the predicted effort of the i^{th} project.

$$MRE(n) = \frac{1}{n}\sum_{i=1}^{n}\left|\frac{x_{act}(i) - x_{pred}(i)}{x_{act}(i)}\right| \tag{1}$$

The *CC* between the actual and predicted series, described by equation (2), measures the ability of the predicted samples to follow the upwards or downwards of the original series as it evolves in the sample prediction sequence. An absolute *CC* value equal or near 1 is interpreted as a perfect follow up of the original series by the forecasted one. A negative *CC* sign indicates that the forecasting series follows the same direction of the original series but with negative mirroring, that is, with a rotation about the time-axis.

$$CC(n) = \frac{\sum_{i=1}^{n}\left[\left(x_{act}(i) - \bar{x}_{act,n}\right)\left(x_{pred}(i) - \bar{x}_{pred,n}\right)\right]}{\sqrt{\left[\sum_{i=1}^{n}\left(x_{act}(i) - \bar{x}_{act,n}\right)^2\right]\left[\sum_{i=1}^{n}\left(x_{pred}(i) - \bar{x}_{pred,n}\right)^2\right]}} \tag{2}$$

The *NRMSE* assesses the quality of predictions and is calculated using the *Root Mean Squared Error* (*RMSE*) as follows:

$$RMSE(n) = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left[x_{pred}(i) - x_{act}(i)\right]^2} \tag{3}$$

$$NRMSE(n) = \frac{RMSE(n)}{\sigma_{\Delta}} = \frac{RMSE(n)}{\sqrt{\frac{1}{n}\sum_{i=1}^{n}\left[x_{act}(i) - \bar{x}_n\right]^2}} \tag{4}$$

If *NRMSE*=0 then predictions are perfect; if *NRMSE*=1 the prediction is no better than taking x_{pred} equal to the mean value of *n* samples.

The Prediction Level (*Pred(l)*) defined in equation (5) specifies how many data predictions *k* out of *n* (total number of data points predicted) performed well, i.e., below a predefined

level specified by the *RE* metric (see equation (6)) is lower than level *l*. In the experiments the parameter *l* was set equal to 0.25.

$$\Pr ed(l) = \frac{k}{n} \tag{5}$$

$$RE(n) = \frac{\left| x_{act}(i) - x_{pred}(i) \right|}{x_{act}(i)} \tag{6}$$

The *Sign Predictor* (*Sign(p)*) metric assesses if there is a positive or a negative transition of the actual and predicted effort trace in the projects used only during the evaluation of the models with the sliding-window technique on unknown test data. With this measure we are not interested in the exact values, but only if the tendency of the next value to the previous is similar. This is expressed in equations (7) and (8).

$$Sign(p) = \frac{\sum_{i=1}^{n} z_i}{n} \tag{7}$$

$$where\ z_i = \begin{cases} 1 & if\ ((x_{t+1}^{pred} - x_t^{pred}) * (x_{t+1}^{act} - x_t^{act})) > 0 \\ 0 & otherwise. \end{cases} \tag{8}$$

4. Experimental approach

As we have already mentioned, the software cost estimation literature has shown many research attempts focusing on predictions of Artificial Neural Network (ANN) models, which are treated as promising estimators with equal or better performance compared to other popular approaches, like by-analogy or regression-based estimation. Their input variables usually involve numerous internal and external project attributes, typically concerning the actual product under development, the people undertaking the development tasks and the process followed.

In our approach size-based data of various size definitions is used, which have been gathered from industrial projects, either representing software actual lines of code or functionality delivered. We investigate cost estimators in the form of ANN models, that aim to learn and generalise the knowledge embedded in past project samples, so as to estimate the associated development effort as accurately as possible. Consequently, our focus is twofold: Firstly, we will study performance, stability and calibration issues of the proposed models and secondly, identify any present correlations of development effort and size-based attributes.

In this section we provide the detailed experimental process and the results yielded by the models developed: (i) An ANN approach with random holdout samples for validation, (ii) An ANN approach, with varying input method (a random timestamp was given to the data samples which were inputted using a sliding-window technique); (iii) A Hybrid model, coupling ANN with a GA to reach to a near to optimal input method and internal architecture; (iv) A classic Regression model.

4.1 An ANN investigating size-effort relation

The following section presents the ANN model proposed to investigate the relationship between software size (expressed in LOC or FP) and effort, by conducting a series of experiments. We are concerned with inspecting the predictive ability of the ANN with respect to the attribute counting the size of the software developed for each project in each dataset.

4.1.1 Model description

ANN are non-linear, model-free and alternative to traditional statistical methods able to solve complex pattern recognition problems. ANN consist of basic computational elements called neurons organised in groups that form layers. They may be also viewed as directed graphs, composed of nodes and connections, also called weights or synapses, which connect the neurons (Haykin, 1999). Certain types of neurons organised in multiple layers form the Multi-Layer Perceptron (MLP) (McCulloch & Pitts, 1943) which is one of the most popular types of ANN. A simple MLP ANN is shown in Figure 1.

The number of neurons in the input (first) layer is equal to the number of attributes used as independent variables. The last layer is the network output which corresponds to the independent variable (in our case software effort). Each subsequent layer uses the weights coming from the previous layers and adjusts them so that the accuracy error between the actual and predicted values for the dependent variable is diminished. Each neuron uses the respective input vectors, the weights and a momentum coefficient to calculate its output.

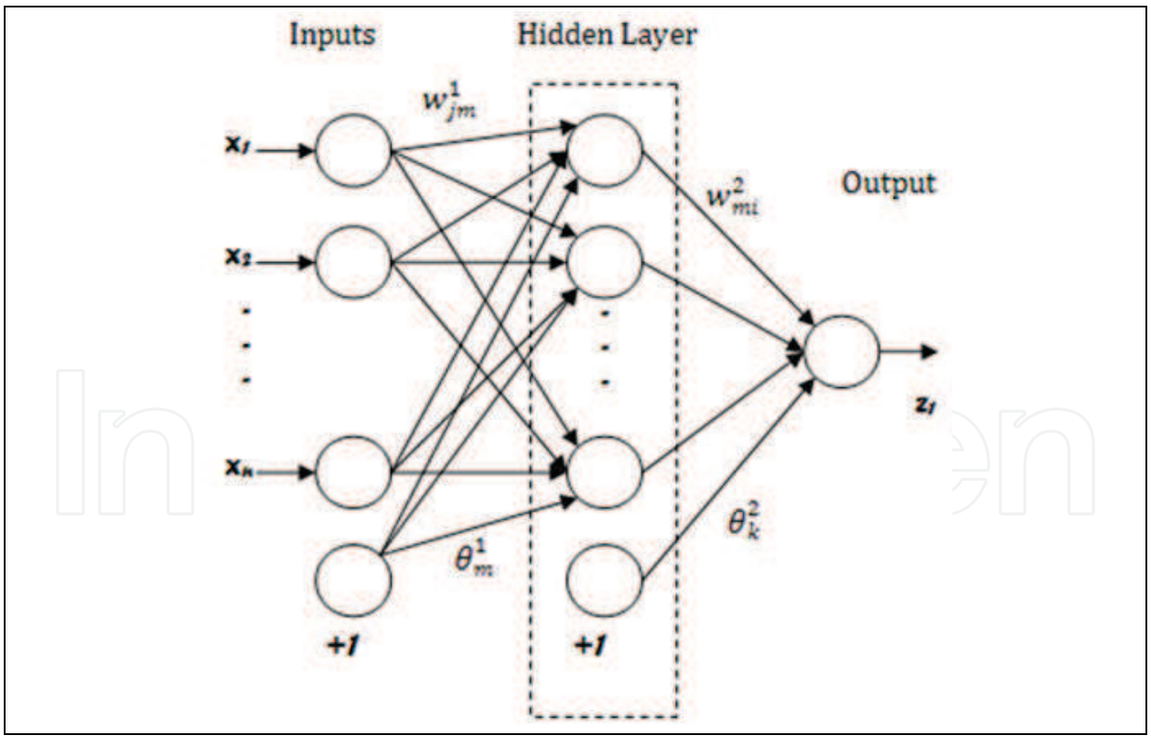


Fig. 1. A feed-forward Multi-Layer Perceptron Neural Network

Equation (9) specifies how the outcome of the first hidden node in the first layer x_1^1 is estimated.

$$x_1^1 = f\left(\sum_{i=1}^{n} x_i w_{i1}^1 + \theta_1^1\right) \tag{9}$$

The networks employed in the present work are single hidden layer networks and MLP networks, which use one hidden layer partitioned into three parallel sub-layers activated by a different function, e.g., the *hyperbolic tangent*, the *Gaussian* and the *Gaussian complement* specified in equations (10), (11) and (12) respectively.

$$f(y) = (1 - \exp(-by)) * (1 + \exp(-by))^{-1} \tag{10}$$

$$f(y) = \exp\left(-x^2\right) \tag{11}$$

$$f(y) = 1 - \exp\left(-x^2\right) \tag{12}$$

The error backpropagation algorithm is one of the most widely used algorithms for training the network and requires data samples in the form of input-output patterns. The backpropagation learning algorithm is used to calculate derivatives of performance of the mean square error with respect to the weight and bias variables. In order to learn efficiently the data fed, the network calculates an error, which is the difference between the desired and the actual response. The error is propagated to the network in a backward manner, so that for each neuron the weights are adjusted to minimise this error iteratively.
Moreover, for the backpropagation algorithm the dataset is randomly divided into three subsets: the training set, the validation set and the testing set. The training set is utilised during the learning process, the validation set is used to ensure that no overfitting occurs in the final result of the learning process and that the network will be able to generalise the knowledge gained. The testing set is an independent subset of the dataset, i.e., does not participate during the learning process and measures how well the network performs with unknown data.

4.1.2 Results

During our experiments we employed a simple, single hidden layer architecture for estimating development effort using LOC or FP from each dataset as input. In case a dataset included both size metrics we developed one ANN model for each metric in order to compare performance results. The number of nodes in the hidden layer was empirically defined for each dataset case due to the simplicity of the models under investigation. For the input layer *netsum* function was used, for the hidden layer the *tansig* function and finally, for the output layer the *purelin* function was used. The models were developed in Matlab R2010b.
Each ANN was trained in a supervised manner, using the backpropagation algorithm and a random selection of 60% of the total projects comprised the training data samples. Also, 20% of the original data samples were used for validation during the training of the ANN and the rest 20% were the holdout samples that were later used for testing the generalisation ability of the best trained model, i.e., the ANN that yielded the lowest *MRE* figure. We randomly initialised the weights and momentum coefficients and re-trained the network 20 times with the backpropagation algorithm. Finally, we utilised the best ANN to proceed to the testing phase.

DATASET	INPUT	TOPO-LOGY	TRAINING PHASE				TESTING PHASE			
			MRE	CC	NRMSE	Pred(.25)	MRE	CC	NRMSE	Pred(.25)
COC`81	LOC	1-4-1	0.888	0.998	0.063	0.333	1.629	0.597	2.890	0.154
KEM`87	FP	1-3-1	0.204	0.966	0.248	0.625	0.282	0.943	0.614	0.333
KEM`87	LOC	1-3-1	0.258	0.861	0.476	0.750	0.257	0.792	0.527	0.333
COKEM`87	LOC	1-2-1	1.335	0.892	0.446	0.132	1.542	0.599	0.971	0.188
ALGAF`83	FP	1-2-1	0.304	0.978	0.213	0.545	0.324	0.987	0.149	0.600
ALGAF`83	LOC	1-4-1	0.301	0.991	0.130	0.364	0.469	0.985	0.691	0.200
DESH`89	FP	1-3-1	0.487	0.697	0.715	0.474	0.348	0.712	0.696	0.400

Table 1. Experimental Results obtained with the ANN-model.

Table 1 summarises the best results obtained with the specific ANN architectures and the various datasets. The first column refers to the dataset used, the second to the type of size metric that was used in the input layer (LOC or FP), the third refers to the ANN topology and the rest of the columns refer to the error metrics during the training and testing phase.

As expected, the degree of accuracy across the datasets varies because accurate and optimum models cannot be developed for every case. However, in some cases the overall performance of the approach is a promising indication that ANN models can reach to quite accurate effort approximations. The datasets whose effort is better approximated is KEM`87, followed by ALGAF`83 using FP as input and then followed by DESH`89. The results of the training phase indicate that the ANN models were able to learn well the training data from all the datasets except COKEM`87, which comprises a concatenation of two different datasets. Therefore, this effect may be attributed to the less homogeneous form of the aforementioned dataset which was merged from two other datasets. The results of the testing phase are also quite successful for KEM`87, ALGAF`83 and DESH`89 datasets but less accurate for the COC`81 and COKEM`87 cases. These figures of effort approximations were considerably easy to achieve, experimenting with only a few internal hidden neurons, i.e., starting from 2 to 5, because we were using one single size attribute as input.

Moreover, for datasets describing both size attributes LOC and FP, i.e., KEM`87 and ALGAF`83, we observe that the ANN models using FP are more accurate in terms of correlation (*CC*) in the first case, and more accurate in prediction level (*MRE*) in the second case, during the testing phase. This indicates that the proposed model can achieve better approximations using the FP size metric instead of LOC, even though it is worth noting that more thorough investigation needs to be performed with different ANN architectures and coupling LOC and/or FP with effort spent on past projects aiming at improving prediction performance.

4.2 An ANN coupling size-effort

The following section presents an ANN model which investigates the relationship between software size (expressed in LOC or FP) and effort, by conducting a series of experiments coupling size and effort data. We are concerned with inspecting the predictive ability of the ANN according to the architecture utilised and the input method (volume and order of the data fed to the model) per dataset.

4.2.1 Model description

The core architecture of a size-effort coupling ANN was a feedforward MLP (as previously described in Figure 1) connecting each input neuron with hidden layers consisting of

parallel slabs activated by different functions. Empirical variations of this architecture were employed regarding the number of inputs and neurons in the internal hidden layers, whereas the difference between the actual and the predicted effort was again manifested at the output layer (forecasting deviation). Again, the ANN were trained in a supervised manner, using the backpropagation algorithm and 70% of the training data samples. Also, 20% of the original data samples were used for validation of the training of the ANN and finally, 10% holdout samples were used for testing the model.

4.2.2 Results

The empirically conducted experiments investigated mainly the appropriate number and type of inputs and internal neurons forming the layers of the ANN. Here, a more complex ANN architecture was used with three different hidden slabs in the internal layers. In addition, in these experiments several ANN parameters were kept constant as some preliminary experiments conducted initially showed that varying the type of the activation function in each layer had no effect on the forecasting quality. More specifically, we employed the following functions: for the input layer the *linear* function [-1, 1], for each respective hidden layer the *Gaussian*, the *tanh*, and the *Gaussian complement* and finally, for the output the *logistic* function.

In addition, for each experiment performed, a sliding-window technique was applied on the randomly generated subsets of training to extract the input vector and feed it to the ANN. Therefore, the selected projects were manifested to the ANN with specific order, so that they would couple size and effort information of past projects developed. Practically, this is expressed in Table 2, covering the following Input Methods (IM) of a varying length (or size *i*) sliding-window, with $i=1,\ldots,5$:

- IM1-IM2: Using the Lines of Code or the Function Points of the i^{th} projects we estimate the effort of the i^{th} projects;
- IM3-IM4: Using Lines of Code or Function Points with effort of the i^{th} project we estimate the effort required for the next project $(i+1)^{th}$ in the series sequence;
- IM5-IM6: Using Lines of Code or Function Points of the i^{th} and $(i+1)^{th}$ projects and effort of the i^{th} project we estimate the effort required for the $(i+1)^{th}$ project.

In each input method the number of past samples included in the sliding-window, that is, the size *i* of the window, is specified from 1 to 5 (*i* index). These combinations enabled us to draw conclusions regarding the dependent variable (effort) for each coupling of input cost drivers and identify its ability to approximate the effort value.

INPUT METHOD	**SOFTWARE METRICS***	**Output***
IM1	$LOC(t_i)$	$EFF(t_i)$
IM2	$FP(t_i)$	$EFF(t_i)$
IM3	$LOC(t_i)$, $EFF(t_i)$	$EFF(t_{i+1})$
IM4	$FP(t_i)$, $EFF(t_i)$	$EFF(t_{i+1})$
IM5	$LOC(t_i)$, $LOC(t_{i+1})$, $EFF(t_i)$	$EFF(t_{i+1})$
IM6	$FP(t_i)$, $FP(t_{i+1})$, $EFF(t_i)$	$EFF(t_{i+1})$

Table 2. Sliding-window technique to determine the ANN input method (*where $i=1,\ldots,5$)..

The best results obtained with the ANN model and the various datasets are summarised in Table 3. The first column refers to the dataset used, the second to the input method with which

the *i* data are fed to the model, the third refers to the ANN topology and the rest of the columns refer to the error metrics during the training and testing phase. The last two columns indicate the number of predicted projects that have the same sign tendency, in the sequence of the effort samples and the total percentage of the successful tendencies during testing.
The figures in Table 3 show that an ANN model deploying a mixture of architectures and input methods yields various accuracy levels. More specifically, the DESH`89 dataset achieves high prediction accuracy, with lowest *MRE* equal to 0.05 and *CC* equal to 1.0. The KEM`87 dataset also performs adequately well with relatively low error figures. The worst prediction performance is obtained with ALGAF`83 and COKEM`87 datasets. These failures may be attributed to the extremely small number of projects involved in the prediction in the first case, and to the use of a heterogeneous dataset in the latter case. Finally, in comparison with the simple ANN models developed previously, the overall prediction accuracy yielded is higher for COC`81, KEM`87 and COKEM`87 datasets, but considerably lower for ALGAF`83 and DESH`89 datasets.

DATASET	INPUT	TOPOLOGY	TRAINING PHASE			TESTING PHASE			Sign (p)	Sign (p) %
			MRE	CC	NRMSE	MRE	CC	NRMSE		
COC`81	IM5	3-15-15-15-1	0.929	0.709	0.716	0.551	0.407	0.952	5/10	50
COC`81	IM1	2-9-9-9-1	0.871	0.696	0.718	0.525	0.447	0.963	7/12	58.33
KEM`87	IM1	1-15-15-15-1	0.494	0.759	0.774	0.256	0.878	0.830	2/3	66.67
KEM`87	IM5	5-20-20-20-1	0.759	0.939	0.384	0.232	0.988	0.503	2/2	100
COKEM`87	IM3	8-20-20-20-1	5.038	0.626	0.781	0.951	0.432	0.948	3/8	37.50
COKEM`87	IM3	4-3-3-3-1	5.052	0.610	0.796	0.768	0.257	1.177	4/8	50
ALGAF`83	IM6	5-3-3-3-1	0.371	0.873	0.527	1.142	0.817	0.649	3/4	75
ALGAF`83	IM2	2-20-20-20-1	0.335	0.975	0.231	1.640	0.936	0.415	2/4	50
DESH`89	IM4	4-9-9-9-1	0.298	0.935	0.355	0.481	0.970	0.247	17/20	85
DESH`89	IM4	6-9-9-9-1	0.031	0.999	0.042	0.051	1.000	0.032	20/20	100

Table 3. Experimental Results obtained with the ANN-model coupling size-effort metrics.

In addition, as the results listed suggest, the COC`81, KEM`87 and DESH`89 datasets achieve adequately fit predictions and thus for some cases, the method is able to approximate the actual development cost. Another observation is that the majority of the best yielded results employ a large number of internal neurons. Therefore, further investigation is needed with respect to different ANN topologies and Input Methods (IM) for the various datasets. To this end, we resorted to using a hybrid scheme, combining ANN with GA, the latter attempting to evolve and reach to the near to optimal network topology and input schema that yields accurate predictions and will have **a** reasonably small size (i.e., number of neurons) so that the computational cost will not radically increase.

4.3 A hybrid ANN & GA

The rationale behind this attempt was that the performance of ANN obtained thus far highly depended on the size, structure and connectivity of the network and results may be further improved if the right ANN configuration parameters are found. Therefore, we applied a GA to investigate whether we can find the ideal network settings by means of a cycle of generations including candidate solutions that are pruned by the criterion 'survival of the fittest', meaning the best performing ANN in terms of effort prediction accuracy.

4.3.1 Model description

The following steps were employed for the Genetic Algorithm implementation:

1. The initial population of individuals was created randomly containing an encoding of the necessary pieces of information, that is, the number of internal hidden neurons and the Input-Method (IM).
2. From each individual of the generation we extracted the information regarding the network architecture and the structure of the input vector. Then the corresponding network was initialised, trained for a number of epochs and finally, simulated. From the simulation results obtained, all individuals were evaluated and the network state and performance results were stored.
3. Once all individuals of the respective generation have been trained and tested on generalisation, the generation was evaluated as a whole.
4. The top 5% of best individuals were forwarded to the next generation (elitism) and the rest individuals missing to complete the next generation were obtained through reproduction steps applying the selection, crossover and mutation operators. The offsprings produced through these steps replaced their parents in the original population.
5. Steps (2), (3) and (4) were repeated until finally, a predefined number of generations have been reached.

More specifically, the first task for implementing the hybrid model was to determine a type of encoding so as to express the potential solutions. The encoding used was a binary string representing the ANN structure, the internal hidden neurons and the varying input's coupling of effort and size attributes. The inputs were inserted into the ANN models created within the hybrid algorithm following the Input Methods (IM) specified earlier. The number of neurons used in the hidden slabs was restricted not to exceed 20 neurons to avoid building ANN models that would lead to overfitting. The space of all feasible solutions (i.e., the set of solutions among which the desired solution resides) was called the search space. Each point in the search space represents one possible solution. Each possible solution was "marked" by its fitness value, which in our case was expressed by equation (13), minimizing the *MRE* and the overall size of the network, i.e., the total number of internal neurons, to avoid creating overly large and complex networks.

$$fitness = \frac{1}{1 + MRE + size} \tag{13}$$

The GA searches the problem space to locate the best solution among a number of possible solutions. Searching for a solution is then equal to looking for some extreme value (minimum or maximum) in the search space.

The GA developed included three types of operators: selection (roulette wheel), crossover (with crossover rate=0.25) and mutation (with mutation rate=0.01). Selection chooses members from the population of chromosomes proportionally to their fitness and elitism was used to ensure that the best members of each population are always selected for the new population. Crossover adapts the genotype of two parents by exchanging parts of them and creating a new chromosome with a modified genotype. Crossover was performed by selecting a random gene along the length of the chromosomes and swapping all the genes after that point. Finally, the mutation operator simply changes a specific gene of a selected individual in order to create a new chromosome with a different genotype.

4.3.2 Results

In this section we present and discuss the results obtained using the Hybrid model on the various available datasets. The best ANN architectures yielded are listed in the third column of Table 4 with the various error figures obtained both during the training and the testing phase.

The performance of the different ANN architectures constructed with the aid of the GA shows high learning ability. The main observation is that for all of the datasets the hybrid model was able to optimise prediction accuracy. This is remarkably consistent through both the training and the testing error figures reported by the best solutions summarised in Table 4. In fact, for all the datasets investigated, the hybrid model performs adequately well in terms of generalisation ability and prediction accuracy. During training and testing the *MRE* is significantly lowered compared to the results of the experiments conducted with the simple ANN (Table 1) and the empirical coupling ANN (Table 3), the *CC* improves in all cases, whereas the *NRMSE* is also highly improved.

Moreover, it is observed that there is a strong relationship between the success of a particular model and the type of attributes used as inputs. In all datasets, IM1 utilising in each case LOC or FP as inputs, yields the best prediction results compared to IM2 and IM3. Accuracy is usually diminished when adding the effort values in IM2 and in all other cases, except in the dataset DESH`89 case, where IM3 accuracy is considerably improved. This shows that the size metric for all datasets improves effort approximations and that LOC or FP are noticeably highly descriptive factors of effort. We would expect that adding the effort values in the inputs of the ANN models would have improved estimates, but this is not the case due to the existence of some projects outliers in respect to their effort values.

Moreover, it seems that the experiments using KEM`87 showed similar *MRE* and *CC* error figures and an improved *NRMSE* in favor of FP instead of LOC, both during training and testing. The ALGAF`83 dataset showed similar *NRMSE* and *CC* error figures, whereas an improved *MRE* was observed using LOC. Overall, the experiments conducted using one of these two size measures for predicting effort (i.e., in IM1 and IM2) produce superior results consistently throughout all the datasets indicating that both LOC and FP are very good descriptors of effort. Of course this is something that on one hand agrees with what is

DATASET	INPUT	TOPOLOGY	TRAINING PHASE			TESTING PHASE		
			MRE	CC	NRMSE	MRE	CC	NRMSE
COC`81	IM1	1-9-17-10-1	0.004	1.000	0.014	0.003	1.000	0.014
COC`81	IM3	2-20-18-3-1	0.092	0.963	0.270	0.075	0.961	0.278
COC`81	IM5	3-19-20-4-1	0.043	0.990	0.149	0.044	0.981	0.199
KEM`87	IM1	1-17-13-16-1	0.008	1.000	0.015	0.009	1.000	0.019
KEM`87	IM3	2-18-14-18-1	0.246	0.825	0.539	0.211	0.822	0.550
KEM`87	IM5	3-19-15-20-1	0.004	1.000	0.006	0.028	0.997	0.081
ALGAF`83	IM2	1-17-20-11-1	0.006	1.000	0.005	0.009	1.000	0.006
ALGAF`83	IM4	2-19-1520-1	0.041	0.998	0.074	0.062	0.993	0.122
ALGAF`83	IM6	3-19-9-16-1	0.029	0.999	0.045	0.031	0.999	0.051
DESH`89	IM2	1-13-20-6-1	0.002	1.000	0.008	0.005	1.000	0.024
DESH`89	IM4	2-19-20-8-1	0.087	0.990	0.136	0.163	0.977	0.210
DESH`89	IM6	3-19-11-10-1	0.089	0.990	0.139	0.173	0.975	0.218

Table 4. Hybrid model (coupling ANN and GA) results.

already pointed out by numerous studies in literature and on the other suggests that our model behaves as it should. Also, another observation is that the best accuracy is significantly lowered (error rates are higher) when the effort is given as an additional input to the ANN (in the IM3 or IM4 cases), meaning that the model's ability to capture the correlation between size and effort is decreased. This shows that outlying values for the effort of these projects exist and indicates that some sort of filtering could improve the results. Another observation is that ANN could be therefore used in the case of software cost estimation as a filtering process to eliminate outlying project values. We additionally observe that prediction accuracy is significantly and consistently improved when the LOC or FP of the project whose effort is being predicted, is given to the model (in the IM5 or IM6 cases). Heuristically, this is a logical conclusion as the model in the latter case is fed with information regarding the project's LOC or FP and therefore, the prediction accuracy is enhanced by this additional information. Overall, the proposed model seems to work under these assumptions consistently well.

4.4 Regression investigating size-effort relation

In this section we present the results obtained from a simple Regression so as to provide some comparative assessment of the models proposed thus far. Regression assesses how well the regression line approximates the real effort and it is built using the same samples used in the ANN training and testing phase.

Regression analysis is used to capture and explain the relationship between the size and effort of projects in the form of an exponential function which can be represented by a polynomial transformed to linear using the natural logarithm.

4.4.1 Model description

We denote Y as the dependent variable of the total cost for developing software projects (usually expressed as the effort spent) and X the independent variables representing the size of projects (usually in LOC or FP). Each vector $\{(x_1,y_1), \ldots, (x_l,y_l)\}$ represents a sample of projects, where $x_i \in \Re^n$ and $y_i \in \Re$ for each project i and are used in the regression model defined in (14). We assume that the errors ε_i are independent and have a zero mean. The goal is to find the polynomial coefficients β_0 and β_1 representing the constant and the slope of the regression linear function $f(x_i)$ respectively, the latter being defined in (15).

$$y_i = f(x_i) + \varepsilon_i \tag{14}$$

$$f(x_i) = \beta_0 + \beta_1 x_i \tag{15}$$

In case the relationship between the dependent and independent variables is not linear we assume that a simple transformation such as the logarithmic can be used to estimate a model of the form (16).

$$y_i = \beta_0 + \beta_1 x_i + \varepsilon_i \tag{16}$$

The error function the approach is trying to minimise is based on the least-squares form.

4.4.2 Results

Regression is built under the assumption that the dependent variable (effort) is linearly related with the independent variable(s) (size and/or next effort values). The model produces the slope of a line that best fits the data of the training set and then, during the testing phase, we estimate the value of the dependent variable. We utilise the yielded regression coefficients from the training phase to estimate the values of effort for the testing samples. Finally, we compare the estimated effort values to the actual effort using the performance metrics mentioned above.

The Regression approach is tested on all the datasets as they were originally separated into the training and testing subsets and were used by the ANN in the previous experiments. Thus, a comparison with the initial ANN models built is feasible. The results of Regression indicate average performance for all datasets with accuracy staying lower compared to that of the approaches previously proposed in this work (simple and hybrid ANN). This indicates that the form of the problem is considered quite complex and cannot be easily addressed by a simple form of Regression. However, although the ANN approach demonstrated some significant advantages in terms of prediction performance in the experiments of this work, it doesn't mean that it can replace Regression but should be regarded as a promising approach for these certain circumstances.

DATASET	INPUT	TRAINING PHASE				TESTING PHASE			
		MRE	CC	NRMSE	Pred(.25)	MRE	CC	NRMSE	Pred(.25)
COC`81	LOC	0.608	0.834	0.579	0.233	1.022	0.639	1.045	0.154
KEM`87	FP	0.424	0.764	0.663	0.500	0.196	0.974	0.289	0.667
KEM`87	LOC	0.330	0.776	0.627	0.500	0.234	0.825	0.566	0.667
COKEM`87	LOC	1.082	0.893	0.591	0.053	0.999	0.607	0.791	0.188
ALGAF`83	FP	0.354	0.930	0.472	0.364	0.248	0.969	0.453	0.600
ALGAF`83	LOC	0.408	0.837	0.530	0.364	0.397	0.970	0.556	0.200
DESH`89	FP	0.514	0.543	0.848	0.395	0.311	0.708	0.819	0.467

Table 5. Regression Results.

The main problem of the Regression method yielding mediocre results may be attributed mainly to the method's dependence on the distribution and normality of the data points used and its inability to approximate unknown functions, as opposed to the ability demonstrated by the simple ANN model as well as the hybrid approach with the GA. However, we recognise that in order to comparatively assess the results of a range of models statistical tests, like Wilcoxon's, or t-tests need to be performed to investigate the statistical difference between the errors yielded by the comparative models.

5. Conclusions

In this chapter, we considered the problem of reliable and accurate software cost estimations through computational intelligence techniques. Effective modelling of the relationship between software effort and size has always been a challenge, especially for people involved

in project resource management, due to the high level of complexity and uniqueness of the software projects developed. The majority of existing estimation models and methods fail to reproduce this relationship so as to yield successful development effort approximations, or have difficulties even to converge to suggesting an explicit, measurable and concise set of factors affecting productivity. Nevertheless, there is a large discussion on the relationship of cost factors and effort and since this relationship is the core of any cost model, it is essential to describe it accurately.

Many studies in the software cost estimation literature encourage the use of Artificial Neural Networks (ANN) as cost predictors showing that they may perform better or at least as well as other approaches. Adopting this position, this chapter involved the investigation of building the relationship of size and effort using ANN. Software size obtained from past historical project data has been proposed as one of the most important attributes affecting effort and has been extensively used to build a variety of cost models.

Essentially, this chapter proposed a modelling approach utilising ANN and the most common size-related factors found in benchmark datasets. These factors refer to software Lines of Code (LOC) and Function Points (FP). The basic assumption of this work was that error-free size measurements are available for a number of software projects obtained from a set of past historical project data which are used as inputs for the ANN cost models created. In addition, a sliding-window of variable length was used to extract size-related sample data from the datasets (i.e., LOC or FP counts) targeting at coupling them with effort subsets from previously completed projects. This coupling was realised in the form of training patterns fed to ANN so as to investigate if a modelling relationship between size and effort may be established. Moreover, a Genetic Algorithm (GA) was implemented to undertake the optimisation of the ANN architecture of the core model to reduce the *Mean Relative Error* (*MRE*). The near-to-optimal ANN topologies and type of inputs selected for each dataset were discussed and compared to Regression models built across the same training and testing data samples.

The results obtained with the ANN models indicated that the performance of such a model mainly depends on its architecture and parameter settings, and relying on empirical rules to determine these settings is not the optimal approach. The problem was thus reduced to finding the ideal ANN architecture to formulate a reliable prediction model for software cost estimation. The first experimental results indicated mediocre prediction success, comparable to the simple Regression, except in a few dataset cases. Also, as the combinations of inputs used in the ANN models increased, we observed that designing an appropriate internal ANN architecture to deal with the complexity in each case (i.e., type of attributes and dataset) was a quite difficult task. Common methods, such as empirical or trial-and-error, often run the risk of overlooking more promising architectures and also, as a result, it was considered particularly hard to further optimise the results yielded by the ANN models. In addition, it became evident that there was need for more extensive exploration of solutions in the search space of various topologies and input methods as the results obtained by the investigated ANN models did not converge to a general solution.

Therefore, this chapter introduced a hybrid model consisting of ANN and Genetic Algorithms (GA). The latter evolved a population of networks to select the optimal architecture and inputs that provided the most accurate software cost predictions. The results of this work showed that the ANN approach combined with a GA yields better

estimates than the empirically created ANN and Regression models, something that suggests that the technique is very promising.

The main limitation of this method, as well as any other size-based approach, is that especially LOC size estimates must be known in advance to provide accurate enough effort estimations, which is never the case. Also, there is always the risk that if the same project was counted twice would not give exactly the same size (LOC or FP) or effort measurements, and this basic limitation is usually recognised between practitioners. In addition, these errors in measurements of size and effort should be taken into consideration in any approach used for cost estimation and especially if a large discrepancy between the actual and estimated size is occurring in estimations made in the early project phases.

Another limitation is the lack of a satisfactory volume of homogeneous data, as well as of a clear definition and measurement rules for size units, such as LOC and FP, which result in uncertainty to the estimation process. The software size is also affected by other factors that are not investigated by the models of this chapter, such as the programming language and platform used during development. This means that we have consciously focused only on coding effort, irrespective of the type of software and development method in this work, which accounts for only a percentage of the total effort in software development. Another important limitation related with the technologies used is that the ANNs are considered "black boxes" and so the GA requires an extensive search of the solution space, something which is considered very time-consuming.

In future research steps we will emphasise on other aspects affecting the prediction performance of ANN, i.e., optimising other ANN parameters of different types of ANN, such as activation functions and learning techniques. Also, evolutionary processes on genetic search could help to automate and improve, if not optimise, ANN design required to represent complex behaviours. An evolutionary algorithm thus could be coupled with ANN in other ways, such as: (i) Employing fixed network structures with connection weights under evolutionary control, which includes both supervised learning applications and reinforced learning applications, (ii) Designing the ordering and organisation of the nodes from the input to the output layer of the network, including arrangement of interconnections, (iii) Pre-processing the input types of the training data, which can be also used to reduce the input set by discarding less informative, or descriptive cost drivers for approximating development effort.

Future research steps may also concentrate on ways to improve the performance of the proposed approach, examples of which may be: (i) Study of more factors affecting development effort and their interdependencies, (ii) Further adjustment of the ANN and GA parameter settings, such as modification of the fitness function, (iii) Improvement of the efficiency of the algorithms by testing more homogeneous or clustered data and, (iv) Improvement of the quality of the data to achieve better convergence. Consequently, more experiments and more thorough investigation of the capabilities of the proposed approaches needs to be conducted but there is also the necessity for the consideration of a larger range of cost drivers.

6. References

Albrecht, A. J. (1979). Measuring Application Development Productivity, *Proceedings of the Joint SHARE, GUIDE, and IBM Application Developments Symposium*, pp. 83-92, Monterey CA, October 1979.

Albrecht, A. J., & Gaffney, J. R. (1983). Software Function Source Lines of Code, and Development Effort Prediction: A Software Science Validation, *IEEE Transactions on Software Engineering*, Vol. 9, No. 6, (November 1983), pp. 639-648, ISSN: 0098-5589.

Angelis, L., Stamelos, I., & Morisio, M. (2001). Building A Software Cost Estimation Model Based On Categorical Data, *Proceedings of the 7th International Symposium on Software Metrics*, IEEE Computer Society, ISBN: 0-7695-1043-4, pp. 4-15, London, 4-6 April 2001.

Azzeh, M., Neagu, D., & Cowling, P. I. (2010). Fuzzy Grey Relational Analysis for Software Effort Estimation. *Empirical Software Engineering*, Vol. 15, No. 1, (February 2010), pp. 60-90, ISSN:1382-3256.

Boehm, B. W. (1981). *Software Engineering Economics*, Englewood Cliffs N. J., Prentice-Hall Inc., ISBN: 0130266922. New Jersey.

Boehm, B. W., Abts, C., Clark, B., & Devnani-Chulani, S. (1997). *COCOMO II Model Definition Manual*, Computer Science Department, The University of Southern California, Los Angeles, CA.

Briand, L. C., & Wieczorek, I. (2002). Resource Modeling in Software Engineering, In: *Encyclopedia of Software Engineering* (2nd edition), Editor: J. Marciniak.

Charette, R. N. (2005). Why software fails. *Spectrum IEEE,* Vol. 42, No. 9, September 2005, pp. 42-49, ISSN: 0018-9235.

Desharnais, J. M. (1988). *Analyse Statistique de la Productivite des Projects de Development en Informatique a Partir de la Technique de Points de Fonction.* MSc. Thesis, Montréal (Université du Québec).

Dolado, J. J. (2001). On the Problem of the Software Cost Function, *Information and Software Technology*, Vol. 43, No. 1, January 2001, pp. 61-72.

Fenton, N. E., & Pfleeger, S. L. (1997). *Software Metrics: A Rigorous and Practical Approach*, International Thomson Computer Press.

Haykin, S. (1999). *Neural Networks: A Comprehensive Foundation* (2nd edition), Prentice-Hall, ISBN: 0132-73350-1, New Jersey.

Heiat, A. (2002). Comparison of Artificial Neural Networks and Regression models for Estimating Software Development Effort, *Information and Software Technology*, Vol. 44, No. 15, December 2002, pp. 911-922.

Idri, A., Khoshgoftaar, T. M., & Abran, A.(2002). Can Neural Networks be Easily Interpreted in Software Cost Estimation? Proceedings of the 2002 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE 2002), ISBN: 0-7803-7280-8, Honolulu, HI, USA, 12-17 May 2002, pp. 1162-1167.

Idri, A, Mbarki, S., & Abran, A. (2004). Validating and Understanding Software Cost Estimation Models based on Neural Networks, *Proceedings of the 2004 International Conference on Information and Communication Technologies: From Theory to Applications*, ISBN: 0-7803-8482-2, Damascus Syria, 19-23 April 2004, pp. 433-434.

Jørgensen, M., & Shepperd, M. (2007). A Systematic Review of Software Development Cost Estimation Studies. Software Engineering, *IEEE Transactions on Software Engineering*, Vol. 33, No. 1, January 2007, pp. 33-53, ISSN: 0098-5589.

Kaur, J., Singh, S., Kahlon, K. S., & Bassi, P. (2010). Neural Network – A Novel Technique for Software Effort Estimation. *International Journal of Computer Theory and Engineering*, Vol. 2, No. 1, February 2010, pp. 17-19.

Kemerer, C. F. (1987). An Empirical Validation of Software Cost Estimation Models, *Communications of the ACM*, Vol. 30, No. 5, May 1987, pp. 416-429, ISSN:0001-0782.

Kemerer, C. F. (1993). Reliability of function points measurement: a field experiment. *Communications of the ACM*, Vol. 36, No. 2, February 1993, pp. 85-97, ISSN:0001-0782.

Kumar, K. V., Ravi, V., Carr, M., & Kiran, N. R. (2008). Software Development Cost Estimation using Wavelet Neural Networks. *Journal of Systems and Software*, Vol. 81, No. 11, November 2008, pp. 1853-1867, ISSN: 0164-1212.

McCulloch, W.S., & Pitts, W. (1943) A Logical Calculus of the Ideas Immanent in Nervous Activity. *Bulletin of Mathematical Biophysics*, Vol. 5, pp. 115-133.

Mittas, N., Kosti, M. V., Argyropoulou, V., & Angelis, L. (2010). Modeling the Relationship between Software Effort and Size Using Deming Regression, *Proceedings of the 6th International Conference on Predictive Models in Software Engineering (PROMISE 2010)*, ISBN: 978-1-4503-0404-7, Timisoara Romania, 12-13 September 2010.

Miyazaki, Y. Terakado, Ozaki, K., & Nozaki, H. (1994). Robust Regression for Developing Software Estimation Models. Journal of Systems and Software, Vol. 27, No. 1, October 1994, pp. 3-16, ISSN: 0164-1212.

Moløkken, K., & Jørgensen, M. (2003). A Review of Software Surveys on Software Effort Estimation, *Proceedings of International Symposium on Empirical Software Engineering*, ISBN: 0-7695-2002-2, Rome Italy, 30 September – 1 October 2003, pp. 223–230.

Park, R. E. (1996). Software size measurement: a framework for counting source statements, CMU/SEI-TR-020. *Software Engineering Institute Carnegie Mellon University*. Available from: http://www.sei.cmu.edu/pub/documents/92.reports/pdf/tr20.92.pdf, Accessed Nov, 2007.

Putnam, L. H. (1978). A General Empirical Solution to the Macro Software Sizing and Estimating Problem. *IEEE Transactions on Software Engineering*, Vol. SE-4, No. 4, July 1978, pp. 345-361, ISSN: 0098-5589.

Putnam, L. H., & Myers, W. (1992). *Measures for Excellence, Reliable Software on Time, Within Budget*, Yourdon Press Computing Series, New Jersey.

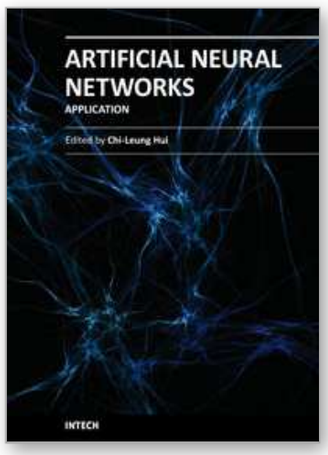
Putnam, L. H., & Myers, W. (2003). *Five core metrics: the intelligence behind successful software management,* Dorset House Publishing, ISBN 0-932633-55-2.

Software Magazine (2004) *Standish: Project success rates improved over 10 years*. Available from: http://www.softwaremag.com/L.cfm?Doc=newsletter/2004-01-15/Standish, Accessed in: November 2007.

Sommerville, I. (2007). *Software Engineering*, Addison-Wesley.

Tronto, I. F. D. B., Silva, J. D. S. D., & Sant'Anna, N. (2008). An Investigation of Artificial Neural Networks based Prediction Systems in Software Project Management, *Journal of Systems and Software*, Vol. 81, No. 3, March 2008, pp. 356-367, ISSN: 0164-1212.

Wittig, G., & Finnie, G. (1997). Estimating software development effort with connectionist models. *Journal of Information and Software Technology*, Vol. 39, No. 7, pp. 469-476.



Artificial Neural Networks - Application

Edited by Dr. Chi Leung Patrick Hui

ISBN 978-953-307-188-6
Hard cover, 586 pages
Publisher InTech
Published online 11, April, 2011
Published in print edition April, 2011

This book covers 27 articles in the applications of artificial neural networks (ANN) in various disciplines which includes business, chemical technology, computing, engineering, environmental science, science and nanotechnology. They modeled the ANN with verification in different areas. They demonstrated that the ANN is very useful model and the ANN could be applied in problem solving and machine learning. This book is suitable for all professionals and scientists in understanding how ANN is applied in various areas.

INTECH
open science | open minds

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821

IntechOpen