Soft Comput (2006) 10: 170–177
DOI 10.1007/s00500-004-0442-z

ORIGINAL PAPER

Xishi Huang · Danny Ho · Jing Ren · Luiz F. Capretz

A soft computing framework for software effort estimation

Published online: 20 April 2005

Abstract Accurate software estimation such as cost estimation, quality estimation and risk analysis is a major issue in software project management. In this paper, we present a soft computing framework to tackle this challenging problem. We first use a preprocessing neuro-fuzzy inference system to handle the dependencies among contributing factors and decouple the effects of the contributing factors into individuals. Then we use a neuro-fuzzy bank to calibrate the parameters of contributing factors. In order to extend our framework into fields that lack of an appropriate algorithmic model of their own, we propose a default algorithmic model that can be replaced when a better model is available. One feature of this framework is that the architecture is inherently independent of the choice of algorithmic models or the nature of the estimation problems. By integrating neural networks, fuzzy logic and algorithmic models into one scheme, this framework has learning ability, integration capability of both expert knowledge and project data, good interpretability, and robustness to imprecise and uncertain inputs. Validation using industry project data shows that the framework produces good results when used to predict software cost.

List of symbols

A_{ik}	Fuzzy set associated with the kth rating level for contributing factor i
ARF_i	Adjusted rating value for contributing factor i
E	Total weighted squared relative errors for all projects
EM_i	Effort multiplier
FM_i	Numerical value for contributing factor i
FMP_{ik}	Parameter value associated with the kth rating level for contributing factor i
$I_{\mathrm{INC}}(F)$	Index set of increasing contributing factors
$I_{\mathrm{DEC}}(F)$	Index set of decreasing contributing factors
M_{o}	Software output metric
$M_{\mathrm{o}n}$	Estimated value of M_{o} for project n
$M_{\mathrm{od}n}$	Desired (actual) value of M_{o} for project n
N	Number of contributing factors
N_{AM2}	Number of non-rating variables in the algorithmic model
N_i	Number of rating levels for contributing factor i
NN	Total number of project data points
NFB	Neuro-fuzzy bank
NFB_i	ith element of the neuro-fuzzy bank
$P=(P_{\mathrm{NFB}}, P_{\mathrm{PNF}}, P_{\mathrm{AM2}})$	The parameter vector including all adjustable parameters in our soft computing framework
	The parameter vector in the neuro-fuzzy bank
P_{PNF}	Parameter vector in the preprocessing neuro-fuzzy subsystem
P_{AM2}	$1\times N_{\mathrm{AM2}}$ parameter vector in the algorithmic model
PFP_{ik}	Parameter value associated with the kth fuzzy if-then rule in PNFIS
PNF, PNFIS	Preprocessing neuro-fuzzy inference system
RF_i	Rating value for contributing factor i
SCF	Soft computing framework

X. Huang · J. Ren · L. F. Capretz
Department of Electrical and Computer Engineering,
University of Western Ontario, London, ON, N6A 5B9, Canada

D. Ho (✉)
Toronto Design Center, Motorola Canada Ltd., Markham,
ON, L6G 1B3, Canada
E-mail: dho27@uwo.ca

SF_i	Scale factor
Size	The size of software project
w_n	Weight of project n
$X = (\mathrm{RF}_1, \mathrm{RF}_2, \ldots, \mathrm{R}F_N, V_1, V_2, \ldots, V_{N_{\mathrm{AM2}}})$	Input vector consisting of contributing factors and non-rating variables
X_n	Vector value of X for project n

1 Introduction

Software development is notorious for going over time and budget and the development cost is difficult to estimate beforehand. This problem lies in the fact that software development is a complex process due to the number of factors involved, including the human factor, the complexity of the product that is developed, the variety of development platforms, and the difficulty of managing large projects. As software development has become an essential investment for many organizations, accurate software cost estimation models are needed to effectively predict, monitor, control and assess software development. Since estimation accuracy is largely affected by modeling accuracy, finding good models for software estimation is now one of the most important objectives of the software engineering community.

Various software cost estimation models (Boehm 1981; Boehm et al. 2000; Putnam and Myers 1992; MacDonell and Gray 1997; Shepperd and Schofield 1997; Chulani 1999) have been developed over the last decades. But because cost estimation problem is a complex problem and has features such as highly complex nonlinear relationships; imprecise and uncertain measurement of software metrics and software processes change rapidly, no model is proved to be the perfect solution so far.

COCOMO (Boehm 1981; Boehm et al. 2000) is arguably the most popular and widely used software estimation model, which integrates valuable expert knowledge. COCOMO, however, has some limitations. It cannot effectively deal with imprecise and uncertain information, and calibration of COCOMO is one of the most important tasks that need to be done in order to get accurate estimations. For example, the COCOMO model can only take on discrete ratings such as six linguistic terms: very low (VL), low (L), nominal (N), high (H), very high (VH) and extra high (XH). This limitation might cause a problem in that the model might produce two quite different cost estimations for two similar projects (e.g. extreme cases: 203 staff-months vs. 2,886 staff-months).

In recent years, some models have been proposed to solve this problem based on neural network or fuzzy logic techniques. Neural networks has learning ability and is good at modeling complex nonlinear relationships; fuzzy logic can deal with imprecise and uncertain measurement of software metrics, provide more flexibility to integrate expert knowledge into the model, and can be easily understood and interpreted. Although these efforts showed a promising research direction in software cost estimation, the approaches based on neural network or fuzzy logic are far from mature.

Idri et al. (2000) introduced the fuzzy logic technique to the COCOMO model. But this fuzzy COCOMO model does not have learning ability, and the results are not good in comparison with the original COCOMO model. Ryder (1998) applied fuzzy modeling techniques to the COCOMO model and the function points model. The fuzzy expert systems are mentioned, but there is no detailed description and no validation results. The fuzzy logic technique is used for software cost estimation (Gray and MacDonell 1997; Pedrycz 2002), but there are no experiments or validation. Wittig and Finnie (1997) developed a software estimation model using neural networks and derived high prediction accuracies. Although their model has high accuracies for its training dataset, the model has not been well-accepted by the community due to its lack of explanation. Neural networks operate as ‘black boxes’ and do not provide any information or reasoning about how the outputs are derived (Idri et al. 2002).

To help solving the software estimation problems, we propose a soft computing framework based on the “divide and conquer” approach. By using a preprocessing neuro-fuzzy inference system to deal with the dependencies among contributing factors, we decouple the effects of contributing factors into individuals, and then we propose a neuro-fuzzy bank for calibrating the factor parameters. The requirements for training project data and computation are greatly reduced, and the fuzzy if-then rules in the neuro-fuzzy bank can remain unchanged for different applications. We validate our soft computing framework with industry project data from several sources and results show that our framework consistently gives good and reasonable performance.

2 A soft computing framework (SCF)

A general problem abstracted from different applications can be formulated as follows: given some information about a project such as ratings of contributing factors, the soft computing framework captures the features of the project, along with the built-in knowledge learned from the historical project data and expert knowledge, to output an accurate and reasonable estimation result. The estimation can provide a powerful assistance to software practice.

An intuitive attempt to solve that problem is to use a five-layer neuro-fuzzy model for software estimation (Gray and MacDonell 1997). In this model, the inputs are the rating values or measurements of the contributing factors and the output is the estimated software output metric. That is, they use a neuro-fuzzy model to handle the dependencies among contributing factors and to calibrate all the parameters of fuzzy if-then rules. Unfortunately, when it comes to a practical problem, the rule base contains a great number of fuzzy rules that result from all the combination of the rating levels of all contributing factors. For example, COCOMO II is arguably the most popular and widely used software estimation model. COCOMO II has 22 cost drivers, and each cost driver has four to six rating levels. If we use the five-layer neuro-fuzzy model to estimate the software cost, we need about 4^{22} to 6^{22} fuzzy rules to deal with all combinations, in theory.

As a result, this approach causes the following problems: (1) First, it needs a very large historical project data set to calibrate the model parameters because the number of project data should be at least greater than the number of fuzzy rules. Unfortunately software project data sets are usually scarce and small. This approach is thus not practical in many cases. (2) The computation will go very high. (3) It is difficult for users, even experts, to check and validate the calibrated fuzzy rules. (4) This approach is not flexible for different applications because the change of application will result in re-definition of all the fuzzy rules. Therefore, this approach is not practical for many software estimation problems, especially for high dimension problems.

For many software estimation problems, the effects of contributing factors on the estimated software output metrics are usually not strongly coupled together, and strong effect dependency exists only on a small number of combinations of contributing factors. Most combinations have no dependency or weak dependency. Consequently, given a specific software estimation problem, we can find the combinations of contributing factors with strong effect dependency based on expert knowledge and analysis of the characteristics of the problem, and then only handle dependency on this small part of combinations. This will greatly reduce the complexity of the estimation problem and make the problem tractable.

The framework described in this paper is a general form for software estimation. It is applicable to various applications such as cost estimation, quality estimation, risk analysis, etc. In this framework (see Fig. 1), we first propose a preprocessing neuro-fuzzy inference system (PNFIS) to handle the dependencies among contributing factors. Then we use a neuro-fuzzy bank to map the adjusted rating values of contributing factors into the corresponding numerical multiplier values. Finally, we output the estimation through an algorithmic model such as COCOMO.

In the following sections, we introduce the architecture of the framework and a detailed learning algorithm. There are three major components in our soft computing framework: preprocessing neuro-fuzzy inference system (PNFIS), neuro-fuzzy bank (NFB) and algorithmic model, as depicted in Fig. 1. The symbols used in this paper are explained in list of symbols.

This architecture is applicable to various applications and allows the utilization of more information from different sources such as expert knowledge and project data. The change of the application will only cause the redefinition of the fuzzy if-then rules in the first part (PNFIS) and the replacement of the algorithmic model in the third part; the rest will remain the same. The expert knowledge on contributing factor dependency is encoded in the first part: preprocessing neuro-fuzzy inference system. The conventional algorithmic model reflects important knowledge on a specific software estimation problem. We use the neuro-fuzzy bank to calibrate the parameters of the algorithmic model through learning from industry project data. Thus, we can integrate the numerical project data, expert knowledge, and conventional algorithmic model into one framework. According to information theory, this process should yield more accurate estimation results. Next, we will introduce the three major components individually in detail.

2.1 Preprocessing neuro-fuzzy inference system (PNFIS)

In many model-based approaches, it is assumed that the effects of contributing factors on the estimated software output metric are independent, but this assumption is often arguable. Even for some very popular and widely used models such as COCOMO, experts diverge from this assumption. However, there is one thing we can claim: this assumption does not hold for all models in all situations. When this assumption breaks down, properly handling of the dependencies among contributing factors becomes essential to improve estimation accuracy. Unfortunately, it is not an easy task in most cases; dependency is inherently a very complex problem as we have to consider the effect of any possible combinations. It is common sense that when it comes to a very complicated situation, experts usually have better understandings of the dependency than any of the advanced computation machines invented so far. Therefore better utilization of expert knowledge is the key to handle this dependency problem. In this paper, we propose the

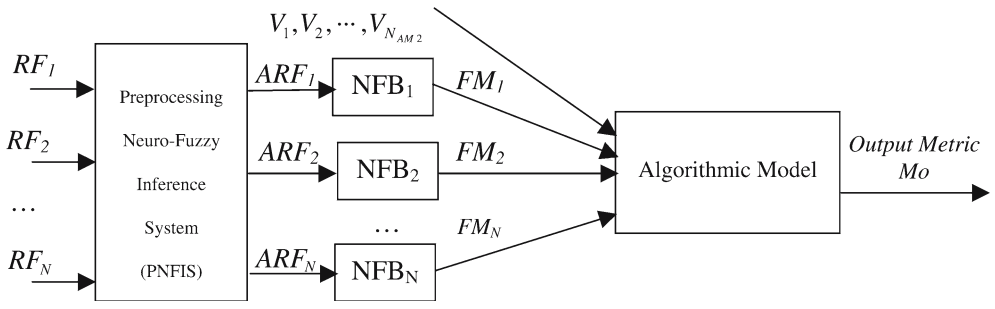

Fig. 1 A soft computing framework for software estimation

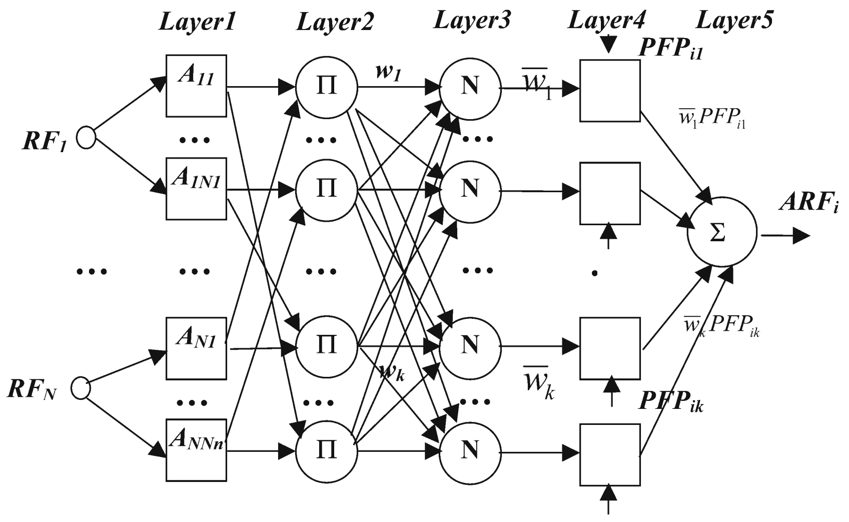

Fig. 2 The structure of PNF$_i$ in PNFIS

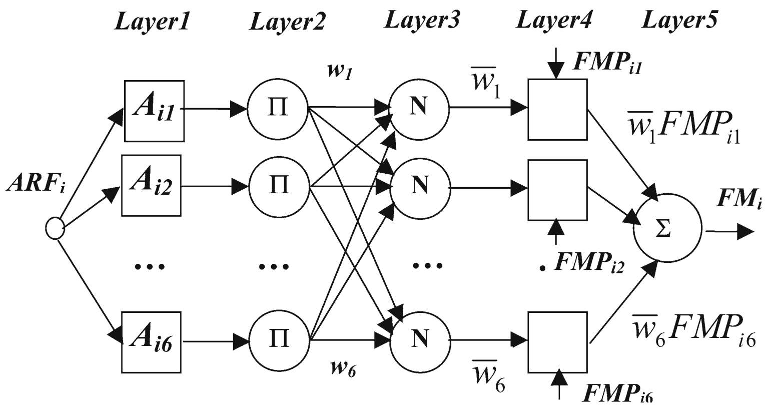

Fig. 3 Element NFB i in the neuro-fuzzy bank

use of a preprocessing neuro-fuzzy inference system (PNFIS) that encodes expert knowledge into fuzzy if-then rules; PNFIS can effectively resolve the effect dependencies among the contributing factors. The inputs of PNFIS are the ratings of the contributing factors (RF$_i$s), and its output is the adjusted ratings of contributing factors (ARF$_i$s), as shown in Figs. 2 and 3.

In PNFIS, fuzzy rules take the form of:

Fuzzy Rule k: IF (RF$_1$ is A_{1jk}) AND (RF$_2$ is A_{2jk}) AND ... AND (RF$_N$ is A_{Njk}) THEN ARF$_i$ = PFP$_{ik}$ × RF$_i$ where PFP$_{ik}$ is an adjustable parameter.

The output ARF$_i$ can be expressed as:

$$\mathrm{ARF}_i = f_{\mathrm{PNF}_i}(\mathrm{RF}_1, \mathrm{RF}_2, \ldots, \mathrm{RF}_N; P_{\mathrm{PNF}}),\quad i = 1, 2, \ldots, N.$$

2.2 The neuro-fuzzy bank (NFB)

In the neuro-fuzzy bank, element i is a neuro-fuzzy subsystem (NFB$_i$), which is associated with contributing factor i. Each contributing factor has several qualitative rating levels, for example, the COCOMO II model uses six rating levels: very low (VL), low (L), nominal (N), high (H), very high (VH) and extra high (XH) for most cost drivers. For a software estimation program, we usually have to define each contributing factor and the corresponding rating criteria for each rating level. But when we use the algorithmic model to estimate the software output metric, we need to use a numerical value corresponding to the rating value of each factor in the mathematical formula. So for every contributing factor, each rating level relates to a quantitative value called parameter value, which is used in the algorithmic model. That is, there is a mapping from rating values to numerical values for each contributing factor. One goal of building a software estimation model is to calibrate its parameter values based on the expert knowledge and project data.

Based on the above features of contributing factors, the structure of each neuro-fuzzy subsystem (NFB$_i$) is chosen as a simplified version of adaptive neuro-fuzzy inference system (ANFIS) (Jang 1993), as shown in Fig. 3. The input of element i is the adjusted rating value ARF$_i$ of contributing factor i; the output is the corresponding numerical value FM$_i$ that is used as the input of the algorithmic model (mathematical formula). For element i of NFB, which is functionally equivalent to a Takaki and Sugeno's type (1985) of the fuzzy system consisting of the following fuzzy rules:

Fuzzy Rule (i,k): If ARF$_i$ is A_{ik}, then FM$_i$ = FMP$_{ik}$, k = 1,2,...,N_i, where N_i is the number of rating levels for factor i, A_{ik} is a fuzzy set associated with the kth rating level of factor i, and FMP_{ik} is the corresponding parameter value associated with the kth rating level. For example, if the contributing factor i has the six rating levels, NFB_i is composed of the following six fuzzy if-then rules:

Fuzzy Rule (i,1): If ARF_i is A_{i1} (very low), then $\mathrm{FM}_i = \mathrm{FMP}_{i1}$,
Fuzzy Rule (i,2): If ARF_i is A_{i2} (low), then $\mathrm{FM}_i = \mathrm{FMP}_{i2}$,
Fuzzy Rule (i,3): If ARF_i is A_{i3} (nominal), then $\mathrm{FM}_i = \mathrm{FMP}_{i3}$,
Fuzzy Rule (i,4): If ARF_i is A_{i4} (high), then $\mathrm{FM}_i = \mathrm{FMP}_{i4}$,
Fuzzy Rule (i,5): If ARF_i is A_{i5} (very high), then $\mathrm{FM}_i = \mathrm{FMP}_{i5}$,
Fuzzy Rule (i,6): If ARF_i is A_{i6} (extra high), then $\mathrm{FM}_i = \mathrm{FMP}_{i6}$.

For each element NFB_i, we also propose a monotonic constraint on the corresponding contributing factor i to guarantee that calibrated results are reasonable. Monotonic constraints reflect the expert knowledge about effects of contributing factors on the estimated software output metric. For most contributing factors, when the rating value of a contributing factor goes high, the estimated software output metric should change monotonically, in other words, it increases or decreases along only one direction, it cannot increase and then decrease or vice versa. Therefore, we can formulate monotonic constraints as follows:

$$\mathrm{FMP}_{i1} \leqslant \mathrm{FMP}_{i2} \leqslant \ldots \leqslant \mathrm{FMP}_{iN_i}, \quad i \in I_{\mathrm{INC}}(F)$$
$$\mathrm{FMP}_{i1} \geqslant \mathrm{FMP}_{i2} \geqslant \ldots \geqslant \mathrm{FMP}_{iN_i}, \quad i \in I_{\mathrm{DEC}}(F)$$

where $I_{\mathrm{INC}}(F)$ is the index set of increasing contributing factors whose higher rating value corresponds to the higher value of the estimated software output metric, and $I_{\mathrm{DEC}}(F)$ is the index set of decreasing contributing factors whose higher rating value corresponds to the lower value of the estimated software output metric. If we do not put monotonic constraints on the neuro-fuzzy bank during the learning process, we might get counterintuitive calibrated results. One advantage of the neuro-fuzzy bank is that we can easily integrate monotonic constraints of contributing factors into our soft computing framework, thus avoid counterintuitive calibrated results. It is much more difficult to place monotonic constraints on a standard five-layer neuro-fuzzy system.

We can look at the functionality of NFB_i from two perspectives. From the learning perspective, we can treat NFB_i as a neural network so that we can use available learning algorithms for neural networks to calibrate the corresponding parameters, and NFB_i has learning and adaptation capability. From the reasoning perspective, NFB_i can be considered as a fuzzy logic system. Its output is derived using fuzzy if-then rules and the reasoning process is transparent in the way similar to the decision-making process of human beings. Therefore, the neuro-fuzzy subsystem NFB_i is not a black box. The whole reasoning process is clear to the user and can be traced and validated by users/experts. Consequently, our neuro-fuzzy model is more easily accepted for project management.

Remark 1: In the neuro-fuzzy bank, each element is a “standard” neuro-fuzzy subsystem. The number of elements is equal to the number of rating contributing factors. In each element, the number of fuzzy rules equals the number of rating levels of the corresponding factor.

Remark 2: For a given problem, once the number of rating contributing factors and the number of rating levels of each factor are determined, the structure and fuzzy rules of the neuro-fuzzy bank are determined. Only the fuzzy rule parameters are needed to fine-tune through the learning process from numerical project data.

Remark 3: Guideline for determining rating levels: how to determine the appropriate number of rating levels. On the one hand, fewer rating levels imply coarse and less accurate description of contributing factors, but the system is easy to use since users can give more accurate rating values. On the other hand, too many rating levels for one factor indicate more accurate descriptions of contributing factors, but the definition and rating criteria of each level are more complicated. Users usually have difficulty rating the factors. That is, the rating errors tend to be larger. Therefore, there is a balance between the number of rating levels and ease of use.

2.3 Algorithmic models

The use of algorithmic models to predict software output metrics is one of the most popular and traditional software estimation techniques. An algorithmic estimation model can be built by analyzing a software output metric and attributes of completed projects, and used to predict the software output metric based on the attributes of software product and development process. Many algorithmic models have been proposed to estimate different software metrics, such as software development cost, software maintenance cost, software quality, and software development productivity. For example, the COCOMO II post architecture model (Boehm et al. 2000) is widely used to predict the software development effort:

$$\mathrm{Effort} = A \times (\mathrm{Size})^{B + 0.01 \times \sum_{i=1}^{5} \mathrm{SF}_i} \times \prod_{i=1}^{17} \mathrm{EM}_i$$

where A and B are constants, size refers to the size of software product, scale factors SF_i’s and effort multipliers EM_i’s are the attributes of software product, platform, personnel and project.

Software development productivity is estimated by the productivity-benchmarking equations (Maxwell 2002):

$$\mathrm{productivity} = A \prod_{i=1}^{N_b} \mathrm{FM}_i$$

where A is a constant, FM_i’s are the contributing factors, N_b is the number of contributing factors, A, FM_i’s, N_b are different for different business sectors.

In order to predict the software maintenance effort, Maxwell proposed an algorithmic model of the form (Maxwell 2002):

$$\text{Maintenance Effort} = A(\text{Size})^B \prod_{i=1}^{4} \text{FM}_i$$

where A and B are constants, FM_i's are the contributing factors.

By analyzing many algorithmic models, we find that many software output metrics take the following forms:

$$\text{Software output metric} = A \prod_{i=1}^{N_b} \text{FM}_i$$

or

$$\text{Software output metric} = A \sum_{i=1}^{N} \text{FM}_i$$

where A is a constant, FM_i's are the major factors that contribute to the software output metric.

For convenience, we call the above models the default algorithmic models in our soft computing framework. If there is no algorithmic model for a given software estimation problem, we suggest to select one of the above default models or its variant as the algorithmic model based on the characteristics of the problem.

Although many algorithmic models are usually not very accurate, they reflect a lot of useful expert knowledge in this field. In order to make the best use of this kind of expert knowledge, it is worthwhile to integrate the algorithmic model into our soft computing framework. Calibrating the parameters of algorithmic models is usually a very challenging task. In our soft computing framework, parameters of the algorithmic model are calibrated by the neuro-fuzzy bank through learning.

3 Software estimation process

The whole estimation process consists of two major steps. In the first step, we calibrate the framework through learning based on historical project data. In the second step, we use the calibrated framework to predict the software output metric of the software product under development and to analyze calibrated parameters for decision-making.

3.1 Reasoning process

Because we adopt the same type of neuro-fuzzy inference system (ANFIS) in both the preprocessing neuro-fuzzy inference subsystem and the neuro-fuzzy bank, for convenience to illustrate the concepts and reasoning process, we consider only the ith element NFB_i in the neuro-fuzzy bank having the rule base:

Fuzzy Rule (i,k): If ARF_i is A_{ik}, then $\text{FM}_i = \text{FMP}_{ik}$, $k = 1,2,\ldots,N_i$

Now we describe the functions of the neuro-fuzzy subsystem layer by layer.

Layer 1: This layer is used to calculate the membership values for each rule. The activation function of a node in this layer is defined as the corresponding membership function:

$$O_k^1 = \mu_{ik}(\text{ARF}_i)$$

where ARF_i is the adjusted rating value of the ith contributing factor, $\mu_{ik}(\text{ARF}_i)$ is the membership function of fuzzy set A_{ik}, which is associated with the kth rating level such as low, high.

Layer 2: This layer calculates the firing strength for each rule. The inputs are the membership values in the premise of the fuzzy rule. The output is the product of all input membership values, which is called the firing strength of the corresponding fuzzy rule. Because there is only one condition in the premise of each fuzzy rule, the firing strength is the same as the membership value obtained from layer 1.

$$w_k = O_k^1$$

Layer 3: This layer is used to normalize the firing strength for each fuzzy rule. The output of the kth node is called the normalized firing strength, which is defined as:

$$\bar{w}_k = \frac{w_k}{\sum_{\text{j}\,=\,1}^{N_i} w_j}, \quad k = 1,2,\ldots,N_i$$

Layer 4: In this layer, the reasoning result of a rule is calculated as follows:

$$O_k^4 = \bar{w}_k \text{FMP}_{ik}$$

Parameters $\{\text{FMP}_{ik}\}$ in this layer are called *consequent parameters*.

Layer 5: This layer sums up all the reasoning results of fuzzy rules we get from layer 4, i.e.,

$$O_k^5 = \sum_k O_k^4.$$

In summary, the overall output of the ith element NFB_i in the neuro-fuzzy bank is

$$\text{FM}_i = \sum_k \bar{w}_k \text{FMP}_{ik} = \sum_{k=1}^{N_i} \frac{\mu_{ik}(\text{ARF}_i)}{\sum_j \mu_{ij}(\text{ARF}_i)} \text{FMP}_{ik}.$$

3.2 Learning algorithms

Given NN project data points $(X_n, M_{\text{od}n})$, $n = 1,2,\ldots,$ NN the learning problem of parameters P in our soft computing framework can be formulated as the following optimizing problem:

$$E = \sum_{n=1}^{\text{NN}} w_n \left(\frac{M_{\text{o}n} - M_{\text{od}n}}{M_{\text{od}n}} \right)^2 \tag{1}$$

subject to the following monotonic constraints:

$$\mathrm{FMP}_{i1} \leqslant \mathrm{FMP}_{i2} \leqslant \ldots \leqslant \mathrm{FMP}_{iN_i}, \quad i \in I_{\mathrm{INC}}(F) \tag{2}$$

$$\mathrm{FMP}_{i1} \geqslant \mathrm{FMP}_{i2} \geqslant \ldots \geqslant \mathrm{FMP}_{iN_i}, \quad i \in I_{\mathrm{DEC}}(F) \tag{3}$$

where E the total weighted squared relative errors for all projects, w_n the weight of project n, $M_{\mathrm{od}\ n}$ the desired (actual) value of the software output metric for project n, X_n the vector value of X for project n.

$$M_{\mathrm{o}n} = f_{\mathrm{SCF}}(X_n; P) \tag{4}$$

$$M_{\mathrm{o}} = f_{\mathrm{SCF}}(X; P) \tag{5}$$

$$\mathrm{ARF}_i = f_{\mathrm{PNF}_i}(\mathrm{RF}_1, \mathrm{RF}_2, \ldots, \mathrm{RF}_N; P_{\mathrm{PNF}}), \quad i = 1, 2, \ldots, N \tag{6}$$

$$\mathrm{FM}_i = f_{\mathrm{NFB}_i}(\mathrm{ARF}_i; P_{\mathrm{NFB}}) = \sum_{k=1}^{N_i} \frac{\mu_{ik}(\mathrm{ARF}_i)}{\sum_j \mu_{ij}(\mathrm{ARF}_i)} \mathrm{FMP}_{ik}, \quad i = 1, 2, \ldots, N \tag{7}$$

$$M_{\mathrm{o}} = f_{\mathrm{AM}}(\mathrm{FM}_1, \mathrm{FM}_2, \ldots, \mathrm{FM}_N, V_1, V_2, \ldots, V_{N_{\mathrm{AM2}}}; P_{\mathrm{AM2}}) \tag{8}$$

The learning algorithm for our soft computing framework is as follows:

$$P_i^{l+1} = P_i^l - \alpha \frac{\partial E}{\partial P_i}, \tag{9}$$

where $\alpha > 0$ is the learning rate, l is the current iteration index,

$$\frac{\partial E}{\partial P_i} = \sum_{n=1}^{\mathrm{NN}} \frac{2w_n}{M_{\mathrm{od}n}^2} (M_{\mathrm{o}n} - M_{\mathrm{od}n}) \frac{\partial M_{\mathrm{o}n}}{\partial P_i} \tag{10}$$

$$\frac{\partial M_{\mathrm{o}n}}{\partial P_i} = \frac{\partial f_{\mathrm{SCF}}}{\partial P_i}, \text{i.e.,} \tag{11}$$

$$\frac{\partial M_{\mathrm{o}n}}{\partial P_{\mathrm{NFB}_i}} = \frac{\partial f_{\mathrm{AM}}}{\partial \mathrm{FM}_i} \cdot \frac{\partial f_{\mathrm{NFB}_i}}{\partial P_{\mathrm{NFB}_i}} \tag{12}$$

$$\frac{\partial M_{\mathrm{o}n}}{\partial P_{\mathrm{AM2}_i}} = \frac{\partial f_{\mathrm{AM}}}{\partial P_{\mathrm{AM2}_i}} \tag{13}$$

$$\frac{\partial M_{\mathrm{o}n}}{\partial P_{\mathrm{PNF}_i}} = \frac{\partial f_{\mathrm{AM}}}{\partial \mathrm{FM}_i} \cdot \frac{\partial f_{\mathrm{NFB}_i}}{\partial \mathrm{ARF}_i} \cdot \frac{\partial f_{\mathrm{PNF}_i}}{\partial P_{\mathrm{PNF}_i}} \tag{14}$$

Remark: Although we derive the learning algorithm for all parameters P, we can calibrate only a part of parameters at one time for a specific software estimation problem.

4 Case studies

In this section, we use data from the software industry to validate our soft computing framework and discuss two case studies.

4.1 Case I. Software cost estimation with the COCOMO model

In this case study, there is a total of 69 project data available, including six project data from the industry (Ho 1996; Panlilio-Yap and Ho 1994) and 63 project data from the original COCOMO'81 database (Boehm 1981). Because most of these project data are compatible only with the intermediate COCOMO'81 model, the algorithmic model employs the COCOMO'81 intermediate model, described by:

$$M_{\mathrm{o}} = \mathrm{Effort} = A_k \times (\mathrm{Size})^{B_k} \times \prod_{i=1}^{15} \mathrm{EM}_i$$

where A_k and B_k are constants specific to each project mode ($k = 1$: organic, $k = 2$: semi-detached or $k = 3$: embedded), EM_is are the cost drivers. In this case, the estimated software output metric M_{o} is software development effort, $\mathrm{FM}_i = \mathrm{EM}_i$, $V_1 = \mathrm{Size}$, $N = 15$, $N_{\mathrm{AM2}} = 1$, that is, the neuro-fuzzy bank has 15 elements, and each element has four to six rating levels. The experiment results (see Table 1) show that our soft computing framework can greatly improve cost estimation accuracy when compared with the standard COCOMO model. Because the COCOMO II model is the more advanced version and has a similar structure to COCOMO'81, we will further validate our soft computing framework when COCOMO II project data become available.

4.2 Case II. Software development effort estimation

With data from 63 projects, Maxwell and Forselius (2000) used the analysis of variance (ANOVA) approach to build multi-variable models to predict the software development effort. In our soft computing framework, we select the 1993 Model B as the algorithmic model, i.e.,

$$M_{\mathrm{o}} = \mathrm{Effort} = A \times (\mathrm{Size})^B \times \prod_{i=1}^{4} \mathrm{FM}_i$$

where A and B are constants, FM_is are the four contributing factors. In this case, the estimated software

Table 1 Effort estimation for all 69 project data points

Predict (%)	COCOMO81 model		Soft computing framework		Improvement (%)
	# Projects	Accuracy (%)	# Projects	Accuracy (%)	
20	49	71	62	89	18
30	56	81	64	92	11
50	65	94	67	97	3
100	69	100	69	100	0

Table 2 Effort estimation for all 63 project data points

Predict (%)	Stepwise ANOVA model		Soft computing framework		Improvement (%)
	# Projects	Accuracy (%)	# Projects	Accuracy (%)	
20	21	33	28	44	11
25	25	39	33	52	13
30	33	52	37	58	6
50	50	79	52	82	3
100	63	100	63	100	0

output metric M_o is software development effort, $N=4$, $N_{AM2}=1$, that is, the neuro-fuzzy bank has four elements, and each element has five or two rating levels. The experiment results are shown in Table 2. Estimation accuracy is not very high due to the low quality of this dataset with many outliers (project data with large noise) in this dataset, however, our soft computing framework improves estimation accuracy a lot when compared with the stepwise ANOVA model.

5 Conclusions

This paper has presented a general framework for software estimation. The framework concentrates on the preprocessing neuro-fuzzy inference system, the neuro-fuzzy bank and the algorithmic model. We consider the rating values of contributing factors as input and produce software metric as output. This framework has been validated with project data from the industry.

The main benefit of this approach is its good interpretability, that is, by using the fuzzy rules, the approach tries to simulate people's line of thought in software estimation. Another great advantage of this research is that we could put together expert knowledge (fuzzy rules), project data and the traditional algorithmic model into one general framework that can have a wide range of applicability in software cost estimation, software quality estimation and risk analysis.

References

Boehm BW (1981) Software engineering economics. Prentice Hall, Englewood Cliffs

Boehm BW et al (2000) Software cost estimation with COCOMO II. Prentice Hall, Englewood Cliffs

Chulani S (1999) Bayesian analysis of software cost and quality models. Doctoral dissertation, Department Computer Science, University of Southern California

Gray A, MacDonell S (1997) A comparison of techniques for developing predictive models of software metrics. Inf Softw Technol 39(6):425–437

Ho D (1996) Experience report on COCOMO and the costar tool from Nortel's Toronto laboratory. In: 11th international forum on COCOMO and software cost modeling, Los Angeles

Idri A, Abran A, Kjiri L (2000) COCOMO cost model using fuzzy logic. In: 7th international conference on fuzzy theory and technology

Idri A, Khoshgoftaar TM, Abran A (2002) Can neural networks be easily interpreted in software cost estimation? In: IEEE international conference on fuzzy systems. pp 1162–1167

Jang RJS (1993) ANFIS: adaptive-network-based fuzzy inference system. IEEE Trans Syst Man Cybern 23:665–685

MacDonell S, Gray A (1997) A comparison of modeling techniques for software development effort prediction. In: International conference on neural information processing and intelligent information systems. Springer, pp 869–872

Maxwell KD (2002) Applied statistics for software engineers. Prentice Hall, Englewood cliffs

Maxwell KD, Forselius P (2000) Benchmarking software development productivity. IEEE Softw 17(1):80–88

Panlilio-Yap N, Ho D (1994) Deploying software estimation technology and tools: the IBM SWS Toronto Lab experience. In: 9th international forum on COCOMO and software cost modeling, Los Angeles

Pedrycz W (2002) Computational intelligence and visual computing: an emerging technology for software engineering. Soft Comput 7:33–44

Putnam L, Myers W (1992) Measures for excellence. Yourdon Press, Englewood Cliffs

Ryder J (1998) Fuzzy modeling of software effort prediction. In: IEEE conference on information technology. pp 53–56

Shepperd M, Schofield M (1997) Estimating software project effort using analogies. IEEE Trans Softw Eng 23(12):736–743

Takaki T, Sugeno M (1985) Fuzzy identification of systems and its application to modeling and control. IEEE Trans Syst Man Cybern 15:116–132

Wittig G, Finnie G (1997) Estimating software development effort with connectionist models. Inf Softw Technol 39:469–476