Improving the Accuracy in Software Effort Estimation

Using Artificial Neural Network Model Based on Particle Swarm Optimization

Zhang Dan
International School
Beijing University of Posts and Telecommunications
Beijing, China
dbb1105@bupt.edu.cn

Abstract—**Recent years, the software industry is growing rapidly and people pay more attention on how to keep high efficiency in the process of software development and management. In the process of software development, time, cost, manpower are all critical factors. At the stage of software project planning, project managers will evaluate these parameters to get an efficient software develop process. Software effort evaluate is an important aspect which includes amount of cost, schedule, and manpower requirement. Hence evaluate the software effort at the early phase will improve the efficiency of the software develop process, and increase the successful rate of software development. This paper proposes an artificial neural network (ANN) prediction model that incorporates with Constructive Cost Model (COCOMO) which is improved by applying particle swarm optimization (PSO), PSO-ANN-COCOMO II, to provide a method which can estimate the software develop effort accurately. The modified model increases the convergence speed of artificial neural network and solves the problem of artificial neural network's learning ability that has a high dependency of the network initial weights. This model improves the learning ability of the original model and keeps the advantages of COCOMO model. Using two data sets (COCOMO I and NASA93) to verify the modified model, the result comes out that PSO-ANN-COCOMO II has an improvement of 3.27% in software effort estimation accuracy than the original artificial neural network Constructive Cost Model (ANN-COCOMO II)**

Keywords—Software project management; software effort estimation model; COCOMO model; particle swarm optimization; artificial neural network

I. INTRODUCTION (*Heading 1*)

With the popularization of information engineering, customers realize the importance of project management and control in software project gradually. This requires accurate enough effort estimate and quality guarantee which provided by the software developers. The development of scale and complexity of the software systems lead to difficulty in estimating software effort, and this may cause an accuracy decrease. What's more, the insufficient and inaccurate in software project effort estimate will result in the failures of lots of software develop project directly. The improvement of the rationality and effectiveness of cost and time allocation in the software development process will promote the efficiency of development significantly. [1] Many project managers tend to use reliable and precise software effort estimation models to evaluate. Among these models, Constructive Cost Model (COCOMO), the effort estimate approach, is commonly acknowledged as the most accurate and easy to use which is based on models and proposed by Barry Boehm in 1981. [2] In 2001, COCOMO II model was proposed which has a significant improvement and modification than COCOMO 81, making it more suitable for estimating the modern software development project. The Post-Architecture Level of COCOMO II includes 17 cost drivers which present software reliability, development tools and programmer abilities, etc. [2] Recent years, artificial neural network is combined with fuzzy system, genetic algorithms and evolutionary computation, and will have a more deep development in practical application. Besides, artificial neural network apply widely in software effort estimate. In [3] and [4], the author provides an effort estimate model which uses artificial neural network. Because it is always estimated and calculated by human judgment, and the software development environment is always different. Besides, software project attributes are usually vague and uncertain. In [5], the author proposes a precise artificial neural network estimation model which incorporates with COCOMO model (ANN-COCOMO II model) to solve the vagueness and uncertainty of the input parameters of COCOMO model. But the shortcomings in this model are the long learning period, the slow convergence rate and the weak ability of network fault tolerance. Particle swarm optimization is a kind of evolutionary computation method based on the swarm intelligence theory, and this algorithm can improve these problems and make the artificial neural network have property of learning and evolution at the same time which shows more intelligent. In [6], an artificial neural network weights learning method is proposed which is under fixed structure and based on particle swarm optimization. This comes out a result that the convergence rate is faster, the network learning accuracy is higher and the performance of learning is better than some common neural networks. In this paper, the optimized algorithm of incorporating the particle swarm optimization with neural network is used, giving an improvement on the model of COCOMO II which is incorporated with artificial neural network, and improving the learnability of neural network to get a more precise effort estimate result.

II. Software Development Effort Estimation Models

A. COCOMO Model

Software project managers and developers prefer to estimate the cost and time consumption at the early phase during the software developing process. The software effort estimate models can be categorized in two classes which are algorithm model and non-algorithm model. The algorithm model is based on the data statistical analysis and historical data, such as software life cycle management (SLCM) and Constructive Cost Model (COCOMO). COCOMO model is regression-based and proposed in 1981 by Boehm which is most cited and the most plausible model in many traditional ones. The improved model COCOMO II which includes software attributes such as 17 Effort Multipliers (EMs), 5 Scale Factors (SFs), Software Size (SS), and Effort estimation are used in the Post Architecture Model of the COCOMO II. [2]

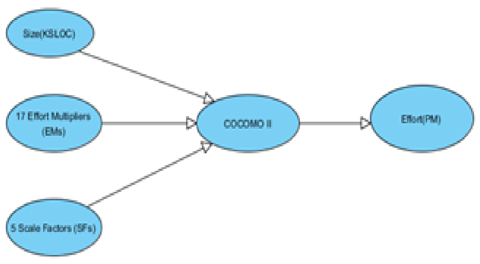

Fig. 1. The constitution of COCOMO II model

The effort in this model is shown in "(1)".

$$\text{Effort(PM)} = \text{A} \times [\text{Size}]^{B} \times \prod_{i=1}^{17} EM_i \quad (1)$$

$$\text{B} = 1.01 + 1.01 \times \sum_{j=1}^{5} Scale\ Factor_j$$

In "(1)":
A: Multiplicative Constant
Size: Size of the software which can be measured in thousands of Source Lines of Code (KSLOC).

Due to the changing cost and different software developing environment at the early phase of software developing process, many software attributes are difficult to obtain. The non-algorithm model which including the soft computation such as neural network, fuzzy logic and evolutionary computation solves this problem.

B. Artificial Neural Network Prediction Model

Because of the self-learning and self-organizing ability to adapt, artificial neural network (ANN) has the characteristics of can be trained. It can absorb experience by learning from the historical data and previous project information which can be used in the new prediction period. Back-propagation algorithm (BP) and feed-forward network are two widely applied ANN estimation technologies. ANN is constituted with active layers and hidden layers, and lots of nodes are connected inside each layer. One connection between two nodes represents a weight and each node represents a special activation function in which sigmoid function is widely used. [7] ANN has the ability of self-learning process, modifying each layer's weight by training samples. The widely used algorithm is BP. In [5], the author proposed a modified ANN model to accommodate COCOMO II post-architecture model. (Fig. 2.)

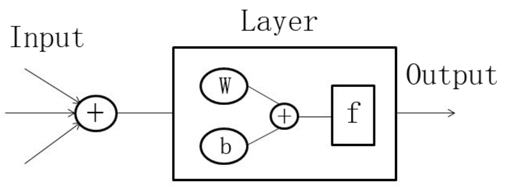

Fig. 2. Structure of proposed ANN model

The new model has 23 input nodes and all of them can satisfy the software size, and scale factors (SFs) and effort multipliers (EMs) are set as two bias variables, and the output of this model is the effort estimate value which has a unit of manpower per month. Besides, this model should consider the data preprocessing, the ANN hidden layer and the sigmoid activation function. [5] In this model, EM_i will be preprocessed as $\log(EM_i)$, and the software size will be computed as thousand lines source code and should be considered as the co-factor of the initial weight in operation. Sigmoid activation function has an expression as $f(x) = \frac{1}{1+e^{-x}}$. Define the relative weight of input nodes P_i as Bias1, the input is $\log(EM_i)$, for $1 \leq i \leq 17$. The weight of scale factors SF_j is $q_j + \log(\text{size})$, for $1 \leq j \leq 5$, and set as Bias2. W and b are presented as the relative weights of the arcs between hidden layer and output layer. This model use BP algorithm as the training method and set the initial values of Bias1 and Bias2 as log(A) and 1.01. Initialize the ANN weights as $p_i = 1$, for , $1 \leq i \leq 17$, and $q_j = 1$, for $1 \leq j \leq 5$. The node values of the hidden layer are showed below:

$$\text{sigmoid}\left(\text{Bias1} + \sum_{1}^{17} p_i * \log(EM_i)\right) = \frac{A * \prod_{i=1}^{17} EM_i}{1 + A * \prod_{i=1}^{17} EM_i} = \alpha \quad (2)$$

$$\text{sigmoid}\left(\log(\text{size}) * \text{Bias2} + \sum_{j=1}^{5} SF_j\right) = \frac{Size^{1.01+\sum_{j=1}^{5} SF_j}}{1 + Size^{1.01+\sum_{j=1}^{5} SF_j}} = \beta \quad (3)$$

And the initialized weight 'W' and 'b':

$$W = \frac{\beta}{2(1-\alpha)(1-\beta)} \quad (4)$$

$$b = \frac{\alpha}{2(1-\alpha)(1-\beta)} \quad (5)$$

The output value of ANN:

$$PM = W * \alpha + b * \beta = \frac{\alpha\beta}{(1-\alpha)(1-\beta)}$$

$$= A.Size^{1.01+\sum_{j=1}^{5} SF_j} * \prod_{i=1}^{17} EM_i \quad (6)$$

The training method is iteration, until the results satisfy the terminal condition which is the change values of all weights less or equal a special threshold. The step of this training algorithm: first, choose the training sample and input the parameters into ANN to get the system output. Second, detect errors of the output and determine the error gradient in all the other layers. Third, determine the changed ANN weights and update them. Finally, redo the steps above until the error obtained from this ANN is small enough and less or equal a special threshold to finish the training period.

But the shortcoming of this model is that the BP algorithm is a training method which is based on error gradient decreasing, and traditional BP algorithm has a slow speed of convergence. The learning performance has a strong dependency on the initial weights of the network. When deal with a nonlinear optimal solution question, it is difficult to find the global optimal solution because of the local limitation. In [6], the author proposed an ANN weight training algorithm which is based on Particle Swarm Optimization (PSO) to solve the problems above. In this paper, this algorithm is used to modify the ANN prediction model and make the effort estimation more precise.

C. Particle Swarm Optimization

Particle swarm optimization (PSO) belongs to evolution computation technology, in 1995, Dr. Eberhaet and Dr. Kennedy proposed it, and this algorithm was derived from researching on the predation of bird flock. Besides, PSO is a research based on population, and this population includes lots of particles where each particle represents a solution of an optimization problem. Like other evolution computation technology, these particles are always initialized randomly. [8] This algorithm optimizes the search of swarm intelligence guidance through the collaboration and competition between particles. The equation of motion of particle i insides the d dimension is showed below.

$$v_{i,d}^{(t+1)} = w \cdot v_{i,d}^{(t)} + c_1 \cdot rand_{id} \cdot \left(p_{i,d}^{(t)} - x_{i,d}^{(t)}\right) + c_2 \cdot rand_{gd} \cdot \left(p_{g,d}^{(t)} - x_{i,d}^{(t)}\right) \quad (7)$$

$$x_{i,d}^{(t+1)} = x_{i,d}^{(t)} + v_{i,d}^{(t+1)} \quad (8)$$

$$\left|v_{i,d}^{(t+1)}\right| \le v_d^{max} \quad (9)$$

- *t*: Iterate Weights, w: inertia factor
- $x_{i,d}^{(t)}$: The particle's current position vector;
- $v_{i,d}^{(t)}$: the particle velocity vector
- $p_{i,d}^{(t)}$: The best location found by particle I (pbest);
- $p_{g,d}^{(t)}$ is defined as the best global position found among all particles in the swarm (gbest);
- c_1, c_2 are accelerated factors;
- $rand_{id}$, $rand_{gd}$ are defend as random numbers uniformly distributed in the interval [0, 1];
- v_d^{max} : The upper limit of the particle speed

The first term on the right hand side of the equation above is corresponding to the momentum which represents the particles' current status. The second and third terms are corresponding to the intensification during the searching process. Particles are moving to the target point by analyzing self-information $p_{i,d}^{(t)}$ and group information $p_{g,d}^{(t)}$ until it satisfied the terminal condition.

The steps of PSO shows below: first, initialize the particle population randomly and initialize the location and velocity of each particle. Second, define the fitness function based on the goal of optimization problem, evaluate the fitness of all particles and update the optimal location of each particle from the population. Third, update the new population optimal location according to the fitness value. Finally, use the functions (7) (8) (9) above and iterate the location and velocity of each particle, redo the steps above until it satisfy the iteration terminal condition of this algorithm and give the output of group optimal location.

However, when solving some complex optimal problems, PSO also has the phenomenon of premature convergence. Hence scholars try to improve the evolutionary mechanism and the optimal performance of PSO recent years. Naka proposed a HPSO method which can be applied into distribution state estimation. It uses the mechanism of natural selection, replacing the particles whose fitness values are low by using those are high in order to make them enter a more efficient searching area. This can estimate the state of target system more precisely by comparing with the traditional PSO algorithm, and even if there may still have some measuring errors, this method can estimate the system state better than others. [9] Huang and Mohan proposed a simple and convenient micro particle swarm algorithm (μPSO), it can be competent a kind of PSO algorithm which has a huge group data by only using a small number of population when considering some high-dimensional optimization problems. Experiments indicate that its calculated amount is less than traditional PSO algorithm significantly when calculating the fitness functions. [10] Ren proposed a method which combines PSO algorithm with BP algorithm, that defines elite particles, and trains the neural network by using a hybrid cross method to give higher accuracy and faster convergence speed of network learning process. [6] This paper give a more precise method by combining the COCOMO II model which is based on ANN and the PSO algorithm (PSO-ANN-COCOMO II) to modify the shortcoming of BP algorithm and PSO algorithm when estimate the software effort. This will be introduced in the next section.

III. The Proposed ANN-COCOMO II Model Incorprates Particle Swarm Optimization

At first, each generation of the particles evolves obeying the PSO algorithm, the updating of each particle's location and velocity can be obtained from the functions above, and also got the fitness value of each particle. And then, based on the fitness value, choose several adaptable particles as elite particles and improve their performance by using BP algorithm. It means that find the locations which can give particles a better performance near the locations of these elite particles by using BP algorithm, and guide faster evolution of next generation particle population. The particles whose performance has been improved constitute a mixed algorithm with the remaining particles of the population, therefore evolves next generation population.

The structure of particle swarm optimization is showed below:

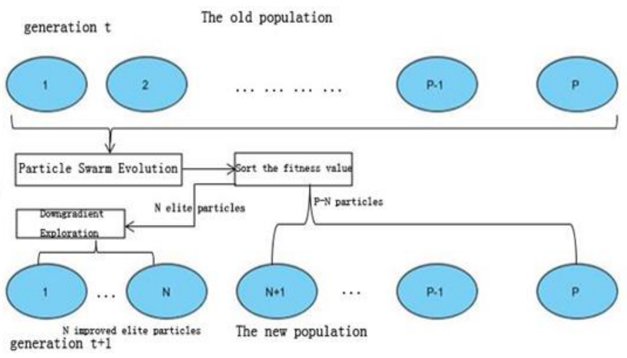

Fig. 3. The structure of particle swarm optimization

Suppose the population scale of particle swarm optimization is P, the number of elite particles is N. When evolve one generation, the training times by using BP algorithm is L, the number of the generations is T. Determine the proportion of elite particles which are used BP algorithm in the population as the algorithm hybrid degree (N/P). When (N/P) = 0, the weights and thresholds of artificial neural network optimize by PSO algorithm. When (N/P) = 1, this network is BP algorithm artificial neural network, the number of training times is T×L. To minimize the calculated quantity, N is usually set less than 10% of the population scale. The pseudo code of the improved algorithm is showed in Fig.4.

This algorithm combined the advantages of particle swarm optimization and BP algorithm efficiently, giving a balance of PSO global exploration and BP local deployment. Using BP algorithm based on PSO algorithm can obtain a more precise optimal solution, and this solution becomes an optimal location (gbest) in the PSO algorithm which can guide the evolvement by using its information. Use BP algorithm in PSO algorithm will increase the calculated quantity of each generation, but this algorithm only consider few elite particles. Handling these particles by BP algorithm, the number of iteration times is small. Hence the increased calculated quantity is small which giving a better performance significantly and improving the data processing efficiency.

The proposed algorithm in pseudo code follows:

```
Initialization population:
if the evolutionary iterations termination criterion is not met then
    for i=1 to Population Size do
        if f($x_i$) < f($p_i$) then
            $p_i = x_i$
            $g_i$ = min ($p_{neighbors}$)
            for d=1 to Dimension do
                $v_{i,d} = w \cdot v_{i,d} + c_1 \cdot rand_{i,d} \cdot (p_{i,d} - x_{i,d}) + c_2 \cdot rand_{g,d} \cdot (p_{g,d} - x_{i,d})$
                $v_{i,d}$ = sign ($v_{i,d}$) ·min(abs($v_{i,d}$) , $v_d^{max}$)
                $x_{i,d} = x_{i,d} + v_{i,d}$
                if $x_{i,d} > x_d^{max}$ or $x_{i,d} < x_d^{min}$ then
                    produce a new particle with randomness to replace the $i^{th}$
                    particle
                end
            end
        end
    end
    Evaluate the population and select N elitists
    for n=1 to N do
        Train the $n^{th}$ particle for local searching with BP operator
    end
end
```

Fig. 4. The pseudo code of the improved algorithm

IV. Result and Discussion

A. Data Sets Description

To match with the original ANN-COCOMO II model, using the data sets below in this paper.

- Data set #1: COCOMO I data set which includes 63 projects data.
- Data set #2: NASA'93 data set which includes 93 small, medium and large scale projects data.

Boehm [1] is the first researcher who found COCOMO I data set and applied it to the software development effort estimation.

B. Evaluation Method

Apply these two data sets to the PSO-ANN-COCOMO II model which is proposed in this paper, and then make a rigorous analysis. Three widely used parameters are used to evaluate the model, which are Magnitude of Relative Error (MRE), Mean Magnitude of Relative Error (MMRE) and Prediction at level L (PRED (L)). MRE is defined as follows:

$$\mathrm{MRE}_i = \frac{|\mathrm{Estimated\ Effort}_i - \mathrm{Actual\ Effort}_i|}{\mathrm{Actual\ Effort}_i} \quad (10)$$

i: project number

The MMRE form project i to project N is defined as follows:

$$\mathrm{MMRE} = \frac{1}{N}\sum_{i}^{N} \mathrm{MRE}_i \qquad (11)$$

Prediction at level L is represented as PRED (L) = k/N which means the probability of a project's relative error less than or equal to level L, where k is the number of projects data whose MRE is less than or equal to L. For example, PRED (25%) means the percentage of the number of projects whose MRE is less than or equal to 25%.

Apply data set #1 and data set #2 on ANN-COCOMO II model and PSO-ANN-COCOMO II model to compute their estimate effort, MRE and PRED (25%), and use MMRE to avoid the occasionality in data processing.

The result which gets from analyzing ANN-COCOMO II model and PSO-ANN-COCOMO II model is showed in TABLE I.

TABLE I. COMPARISON OF THE RESULTS OBTAINED FROM ANN-COCOMO II MODEL AND PSO-ANN-COCOMO II MODEL

Data set	**Model**	**Evaluation**	
		MRE	***PRED(25%)***
Data set #1	ANN-COCOMO II	0.487257017	52%
	PSO-ANN-COCOMO II	0.457260001	55.1%
Data set #2	ANN-COCOMO II	0.428617366	39%
	PSO-ANN-COCOMO II	0.401253700	43.2%
Mean	ANN-COCOMO II	0.457937192	45.5%
	PSO-ANN-COCOMO II	0.434765102	49.5%

TABLE I shows the result after applying the two data sets on ANN-COCOMO II model and PSO-ANN-COCOMO II model, and the third column shows the mean of MRE and PRED (25%). In ANN-COCOMO II model, the means of MRE and PRED (25%) are 0.457937192 and 45.5%, and in PSO-ANN-COCOMO II model, the means of MRE and PRED (25%) are 0.434765102 and 49.5%.

C. *Result Analysis*

- In software development effort estimation model, the smaller MMRE means the better estimation result. The MMRE of PSO-ANN-COCOMO II is 0.434765102 which is smaller than the MMRE 0.457937192 of ANN-COCOMO II model. Hence, the effort estimation is more accurate in PSO-ANN-COCOMO II model than in ANN-COCOMO II model.
- In software development effort estimation model, the bigger PRED (25%) means higher effort estimation accuracy. The PRED (25%) in PSO-ANN-COCOMO II is 49.5% which is bigger than 45.5% in ANN-COCOMO II model. Hence, the effort estimation is more accurate in PSO-ANN-COCOMO II model than in ANN-COCOMO II model.

The conclusion can be obtained from the analysis above that the effort estimation is more accurate in PSO-ANN-COCOMO II model than ANN-COCOMO II model, and the error rate of software effort estimation in PSO-ANN-COCOMO II model is smaller than that in ANN-COCOMO II model.

TABLE II shows the improvement of estimate accuracy between PSO-ANN-COCOMO II model and ANN-COCOMO II model.

TABLE II. IMPROVEMENT IN ESTIMATION ACCURACY OF THE PSO-ANN-COCOMO II MODEL AND ANN-COCOMO II MODEL.

Model	**Evaluation**	
PSO-ANN-COCOMO II vs. ANN-COCOMO II		***MMRE***
	ANN-COCOMO II	0.457937192
	PSO-ANN-COCOMO II	0.433452890
	Improvement%	3.27%

This table shows the comparison of MMRE between PSO-ANN-COCOMO II model and ANN-COCOMO II model. PSO-ANN-COCOMO II model gives an improvement of 3.27% in MMRE which means PSO-ANN-COCOMO II model provides more accurate software effort estimation.

V. CONCLUSION

Time, cost, and manpower are critical factors in software develop process, an effective develop process that can be achieved by evaluating these parameters at the early phase of the project. Software effort evaluation will lead an improvement of software develop efficiency and increase its successful rate. The original ANN-COCOMO II model can solve the problems of vagueness and uncertainty of the software attributes, but it still has some shortcomings such as the low speed of convergence and the high dependency of the network's initial weights. To solve these problems, particle swarm optimization algorithm is imported. This paper proposes the ANN-COCOMO II model which is based on PSO algorithm to optimize the learning attribute. Besides, it keeps the advantages of COCOMO model, improving the accuracy of software effort estimation and decreasing the error rate. The result indicates that, compared with original ANN-COCOMO II model, the accuracy of software effort estimation has increased 3.27% by applying PSO-ANN-COCOMO II model. Some other evolutionary computation technologies such as genetic algorithm can be explored and applied on software effort estimation models in the future.

REFERENCES

[1] B. Boehm, Software Engineering Economics, Prentice-Hall: Englewood Cliffs, NJ, 1981.

[2] B. Boehm, C. Abts, and S. Chulani, "Software Development Cost Estimation Approaches – A Survey," University of Southern California Center for Software Engineering, Technical Reports, USC-CSE-2000-505, 2000.

[3] M. Shepper and C. Schofield, "Estimating software project effort using analogies," IEEE Tran. Software Engineering, vol. 23, pp. 736–743, 1997.

[4] G. Witting and G. Finnie, "Estimating software developemnt effort with connectionist models," in Proceedings of the Information and Software Technology Conference, pp. 469–476, 1997.

[5] Iman Attarzadeh, Amin Mehranzadeh, Ali Barati, "Proposing an Enhanced Artificial Neural Network Prediction Model to Improve the Accuracy in Software Effort Estimation, " in Computational Intelligence, Communication Systems and Networks Conference, pp. 167-172, 2012.

[6] Z.W. Ren, "Research on neural network learning algorithms based on evolutionary computation," Harbin Institute of Technology for Control Science and Engineering, Dissertation for the Doctoral Degree in Engineering, 2008.

[7] L. H. Putnam, "A General Empirical Solution to the Macro Software Sizing and Estimating Problem," IEEE Transactions on Software Engineering, vol. 4, no. 4, 1978, pp. 345 – 361.

[8] Alaa Sheta, David Rine, Aladdin Ayesh, "Development of Software Effort and Schedule Estimation Models Using Soft Computing Techniques," in IEEE Congress on Evolutionary Computation Conference, pp. 1283-1289, 2008.

[9] S. Naka, T. Genji, T. Yura, et al. A Hybrid Particle Swarm Optimization for Distribution State Estimation. IEEE Transactions on Power System. 2003, 18(1):60–68

[10] T. Huang, A. S. Mohan. Micro-Particle Swarm Optimizer for Solving High Dimensional Optimization Problems. Applied Mathematics and Computation. 2006, 181(2):1148–1154