Optimizing Parameters of Software Effort Estimation Models using Directed Artificial Bee Colony Algorithm

Thanh Tung Khuat, My Hanh Le
The University of Danang, University of Science and Technology, Danang, Vietnam
thanhtung09t2@gmail.com, ltmhanh@dut.udn.vn

Keywords: Software effort estimation, directed artificial bee colony, optimization, swarm intelligence, estimation models

Received: December 5, 2016

Effective software effort estimation is one of the challenging tasks in software engineering. There have been various alternatives introduced to enhance the accuracy of predictions. In this respect, estimation approaches based on algorithmic models have been widely used. These models consider modeling software effort as a function of the size of the developed project. However, most approaches sharing a common thread of complex mathematical models face the difficulties in parameters calibration and tuning. This study proposes using a directed artificial bee colony algorithm in order to tune the values of model parameters based on past actual effort. The proposed methods were verified with NASA software dataset and the obtained results were compared to the existing models in other literature. The results indicated that our proposal has significantly improved the performance of the estimations.

Povzetek: S pomočjo algoritma umetne čebelje kolonije so optimirani parametri za oceno potrebnega softverskega dela.

1 Introduction

Software effort estimation is the process of predicting the most realistic amount of effort which is usually expressed in terms of person-hours required to develop or maintain software based on incomplete, uncertain and noisy input. This activity has become a crucial task in software engineering and project management. Effort estimation at the early stages of software development is a challenge due to the lack of understanding the requirements and information regarding to the project. Both underestimated and overestimated effort are harmful for projects under development. Underestimation results in a situation where commitments of the project cannot be accomplished because of a shortage of time and/or resources. In contrast, overestimation can lead to the rejection of a project proposal or cause the allocation of an excess of resources to the project [1].

Several techniques for cost and effort estimation have been proposed over the last few decades, and they can be classified into three main categories [2]. These categories comprise:

1. **Expert judgement** [3]: a technique widely used, relies on an expert's previous experience on similar projects to gather, evaluate, discuss, and analyze data concerning a target project to generate an estimation.

2. **Algorithmic models** [4]: also known as parametric models attempt to represent the relationship between effort and characteristics of project. The main cost driver of these models is the software size, usually measured by Kilo Line of Code (KLOC) or function point. This is still the most popular technique in the literature [26]. These models include COCOMO I [6], COCOMO II [7], SLIM model [8], and SEER-SEM [9].

3. **Machine learning**: In recent years, machine learning approaches have been used in conjunction or as alternative to the above two techniques. These techniques in this group consist of fuzzy logic models [10], neural networks [11], case-based reasoning [2], and regression trees [12].

However, none of the aforementioned approaches are complete and can be appropriate in all situations [13]. This study focuses on the algorithmic models and deals with their difficulties in parameters calibration and tuning in order to enhance the accuracy of software effort predictions. The objective of this study is to focus on improving the algorithmic models which were proposed in the literatures of Sheta [14] and Uysal [15] by using the directed artificial bee colony algorithm with several modifications.

Swarm intelligence is the discipline that collectives behaviors from the local interactions of the individuals with each other and with their environment. This discipline also models swarms that are able to self-organize [16]. Examples of systems studied by swarm intelligence are behaviors of real ants [17], schools of fish, flocks of birds [18]. Artificial bee colony (ABC) algorithm [16] is one of the most-studied swarm intelligence algorithms. There have been a lot of improved variants of ABC algorithm used to tackle a wide range of optimization problems. Among these studies, the directed artificial bee colony (DABC) al-

gorithm [19] is a new version of basic ABC. This algorithm is better than the original ABC in terms of solution quality and convergence characteristics [19]. This work, therefore, applies the DABC in order to optimize parameters of algorithmic models for software cost estimation.

Our contributions in this paper include:

- We combine a control parameter and direction information for each dimension of food source position to update position of the current food source in the state-of-the-art of the Directed Artificial Bee Colony Algorithm. We also use a new boundary constraint-handling mechanism for the DABC if the position of the food source exceeds the boundaries of variables.
- We apply the DABC to improve the effort estimation models introduced in recent literature.
- We evaluate the efficiency of the proposed approaches compared with original models.

The rest of this paper is organized as follows. Section 2 briefly represents software effort estimation models in general and algorithmic models in particular. Section 3 shows the DABC algorithm. Experimental results are presented in Section 4 and Section 5 concludes the obtained results of the study.

2 Software effort estimation models

Estimation methods based on algorithmic models are common. Researchers have attempted to derive algorithmic models and formulas to present the relationship between size, cost drivers, methodology used in the project and effort. As a result, an algorithmic model can be expressed as follows:

$$Effort = f(x_1, x_2, ..., x_n) \quad (1)$$

where $\{x_1, x_2, ..., x_n\}$ denote the cost factors. In additon to the software size, there are many other cost factors proposed and used by Boehm *et al* in the Constructive Cost Model (COCOMO) II [20]. These cost factors can be divided into four categories:

- **Product factors**: required software reliability, database size, product complexity.
- **Computer factors**: execution time constraint, main storage constraint, virtual machine volatility, and computer turnaround constraints.
- **Personnel factors**: analyst capability, application experience, programmer capability, virtual machine experience, and language experience.
- **Project factors**: multi-site development, use of software tool, and required development schedule

The existing algorithmic models differ in two aspects: the selected cost factors, and the function f used.

2.1 Algorithmic models

The simplest formula of relationship between effort and input factors is a linear function, which means that if size increases then effort also rises at a steady rate. Linear models have the form:

$$Effort = a_0 + \sum_{i=1}^{n} a_i * x_i \quad (2)$$

where the coefficients $a_0, a_1, ..., a_n$ are selected to best fit the completed project data.

The linear model, nevertheless, is not appropriate for estimates of non-trivial projects in large and complicated environments. Therefore, more complex models were developed. These ones reflected the fact that costs do not normally increase linearly with project size. In the most general form, an algorithmic estimation for software cost can be represented as:

$$Effort = A * Size^B * M \quad (3)$$

where *A* is a constant factor depending on local organizational practices and the kind of software that is developed. *Size* might be either the size of the software or a functionality estimation expressed in function or object points. The value of exponent *B* is normally between 1 and 1.5. *M* is a multiplier formulated by combining process, product and development attributes.

COCOMO which was introduced by Boehm [21] is one of a very famous software effort estimation models using general formula presented in Eq. 3. The COCOMO model is an empirical model that was derived by collecting data from 63 software projects. These data were analyzed to construct a formula that was the best fit to the observations. The formula of the basic COCOMO is shown in Eq. 4.

$$E = A * (KLOC)^B \quad (4)$$

where *E* shows the software effort computed in person-months. *KLOC* stands for Kilo Line of Code. The values of the parameters *A* and *B* depend mainly on the type of software project. There were three classes of software projects that were classified based on the complexity of projects. They are Organic, Semidetached and Embedded models [21].

1. For simple, well-understood applications (Organic): *A* = 2.4, *B* = 1.05
2. For more complex systems (Semidetached): *A* = 3.0, *B* = 1.15
3. For Embedded systems: *A* = 3.6, *B* = 1.2

COCOMO model ignores requirements and documentation, customer skills, cooperation, knowledge, hardware issues, personnel turnover levels at all. Therefore, with regard to the complex projects, the estimated results using COCOMO model are not accurate. Extensions of COCOMO, such as COMCOMO II [20], enhanced the quality

of software estimates. However, this paper does not take into consideration this model.

Another method to improve the quality of the COCOMO model is to complement a methodology (ME) factor used in the software project into the equation to estimate effort. It was also found that adding the ME factor similar to the classes of regression models assists to stabilize the model and reduce the influence of noise in measurements [14]. Software effort estimation model is changed to:

$$E = f(KLOC, ME) \tag{5}$$

where f is a nonlinear function in terms of KLOC and ME.

Sheta [14] presented two various versions for function f as follows:

– **Sheta's Model 1**:

$$E = A * (KLOC)^B + C * ME \tag{6}$$

where $A = 3.1938, B = 0.8209, C = -0.1918$.

– **Sheta's Model 2**:

$$E = A * (KLOC)^B + C * ME + D \tag{7}$$

where $A = 3.3602, B = 0.8116$,
$C = -0.4524, D = 17.8025$.

Uysal [15] developed Sheta's models and proposed two new models as the following functions:

– **Uysal's Model 1**:

$$E = A * (KLOC)^B + C * ME^D + E \tag{8}$$

where $A = 3.3275, B = 0.8202$,
$C = -0.0874, D = 1.6840, E = 18.0550$.

– **Uysal's Model 2**:

$$\begin{aligned} E = A * (KLOC)^B + C * ME^D \\ +E * ln(ME) + F * ln(KLOC) + G \end{aligned} \tag{9}$$

where $A = 3.8930, B = 0.7923$,
$C = -0.2984, D = 1.3863, E = 2.8935$,
$F = -1.2346, G = 15.5338$.

This study uses the DABC algorithm with several modifications to optimize the parameters of four aforementioned models in order to enhance the accuracy of estimates.

2.2 Measuring estimation quality

The approaches which are widely used to evaluate the quality of software effort estimation models encompass:

– The Mean Magnitude of Relative Error (MMRE) [22]

– The Median Magnitude of Relative Error (MdMRE) [23]

– The Prediction at level *N* (PRED(*N*)) [24]

The Mean Magnitude of Relative Error is probably the most widely employed evaluation criterion for appraising the performance of software prediction models [26]. The MMRE is defined as Eq. 10.

$$MMRE = \frac{1}{T}\sum_{i=1}^{T} MRE_i \tag{10}$$

where T is the number of observations, *i* expresses each observation for which effort is predicted and MRE is Magnitude of Relative Error, which is computed as:

$$MRE_i = \frac{|ActualEffort_i - EstimatedEffort_i|}{ActualEffort_i} \tag{11}$$

Conte *et al.* [24] indicated that $MMRE \leq 0.25$ is acceptable for effort estimation models. Given two data sets A and B, suppose that data set A includes small projects whereas B contains large projects. Given everything else is equal and MMRE(B) is smaller than MMRE(A). As a result, a prediction model assessed on data set B will be considered as better than a competing model evaluated on data set A.

Unlike the mean value, the median always shows the middle value *m*, given a distribution of values, and assures that there is the same number of values above *m* as below *m*. Therefore, the median of MRE values for the number of observations called the MdMRE is an alternative to evaluate the performance of software prediction models. Similar to MMRE, the value of MdMRE less than or equal to 0.25 is acceptable for effort estimation models.

Another method which is commonly used is the Prediction at level *N* known as PRED(*N*). It is the percentage of projects for which the predicted values fall within $N\%$ of their actual values. For instance, if PRED(25) = 85, this indicates that 85% of the projects fall within 25% error ranges. Conte *et al* [24] claimed that *N* should be set at 25% and a good estimation system should offer this accuracy level to 75% of the effort. Eq. 12 illustrates the way to compute the value of PRED(*N*):

$$PRED\,(N) = \frac{100}{T} * \sum_{i=1}^{T} \begin{cases} 1, & \text{if } MRE_i \leq \frac{N}{100} \\ 0, & \text{otherwise} \end{cases} \tag{12}$$

Although MMRE and MRE were frequently used for assessing the accuracy of effort estimation, Shepperd and MacDonell [25] criticized that the use of these criteria is biased. For instance, we have two projects where the first project is an over-estimate and the second project is an under-estimate. The actual and estimated values of the effort of project 1 are 20 and 100 respectively. Project 2 has the actual effort value being 100, and the estimated value is 20. Both estimates have identical absolute residual with 80, but the MMRE values differ by an order of magnitude. Consequently, MMRE will be biased towards prediction systems that under-estimate [25]. Therefore, Shepperd and MacDonell proposed a novel measure called mean absolute

residual (MAR), and it is shown in Eq. 13.

$$MAR = \frac{\sum_{i=1}^{T} |ActualEffort_i - EstimatedEffort_i|}{T} \quad (13)$$

This paper uses all four approaches above to assess the accuracy of the various software effort estimation models presented. Next section shows the DABC algorithm with some modifications to optimize the parameters of prediction models.

3 Directed artificial bee colony algorithm

The original ABC algorithm was proposed by Karaboga [16] based on simulating intelligent behavior of real honey bee colonies. One half of the artificial bees population contains the employed bees, while the other one includes the onlookers and scouts. In this algorithm, the total number of food sources (solutions) is equal to number of employed bees. The employed bees search the food around the food source and then give their information about the quality of the food sources to the onlookers. On the basis of information obtained, the onlooker bees make a decision, which food source to visit, and further search the foods around the chosen food sources. When the food source is exhausted, the corresponding employed and onlooker bees become scouts. These bees will abandon the food source and search for a new food source randomly. The general structure of ABC algorithm is shown in Algorithm 1.

Algorithm 1 General framework of Artificial Bee Colony Algorithm

```
Initialization Phase
while Cycle < Maximum Cycle Number (MCN) do
    Employed Phase
    Onlooker Phase
    Scout Phase
    Memorize the best solution achieved so far
end while
```

There are four control parameters in the original ABC. They are the maximum cycle number (MCN), the size of the population (SP) (the sum of numbers of employed and onlooker bees), the number of trials for abandoning food source *limit* and the number of scout bees (usually chosen as 1) [16].

In the initialization phase, the population of solutions is randomly produced in the range of parameters by using Eq. 14:

$$x_{ij} = Lb_j + r * (Ub_j - Lb_j), \quad i = 1, 2, ..., SP/2,\ j = 1, 2, ..., D \quad (14)$$

where *D* is the number of decision variables of the problem (also the number of variables need to be optimized in software effort estimation models), x_{ij} is the j^{th} dimension of the i^{th} food source which will be assigned to the i^{th} employed bee. Lb_j and Ub_j are the lower and upper bounds of the j^{th} dimension respectively, *r* is a random number in the range of [0, 1].

After the food sources are generated, their qualities are measured by using Eq. 15.

$$fit_i = \begin{cases} \frac{1}{1+f_i}, & \text{if } f_i > 0 \\ 1 + |f_i|, & \text{otherwise} \end{cases} \quad (15)$$

where fit_i is the fitness of the i^{th} food source, and f_i is the corresponding cost function value for the optimization problem. This study uses f_i as the value of MMRE on N projects of the training dataset using parameter values of the i^{th} food source, $f_i = MMRE_i$.

In the employed bee phase of the original ABC algorithm, every solution x_i (i = 1,...,SP/2) is updated according to the following equation:

$$v_{ij} = x_{ij} + \varphi * (x_{ij} - x_{kj}) \quad (16)$$

where v_i is the candidate food source position generated for food source position x_i, x_{ij} denotes the j^{th} parameter of x_i, *j* and *k* are random indexes ($j, k \in \{1, ..., SP/2\}$), φ is a uniform random number in the range of [-1, 1], x_k presents the other solution chosen randomly from the population.

It can be seen that only one dimension of the food source position is updated by the employed bees. This leads to a slow convergence rate. In order to overcome this issue, Akay and Karaboga [27] introduced a control parameter called modification rate (MR). In this improved algorithm, whether a dimension will be updated is decided by using the predefined MR value which is a number in the range of [0,1]. Eq. 16 is modified as follows:

$$v_{ij} = \begin{cases} x_{ij} + \varphi * (x_{ij} - x_{kj}), & \text{if } r_{ij} < MR \\ x_{ij}, & \text{otherwise} \end{cases} \quad (17)$$

where r_{ij} is a random number generated in the range of [0, 1] for the j^{th} parameter of x_i. If r_{ij} is less than MR, the dimension j is changed and at least one dimension is updated by using Eq. 16, otherwise the dimension j is remained [27].

Kiran *et al.* [19] claimed that the search process around the current food source in the basic ABC is fully random in terms of direction because φ is a random number in [-1, 1]. Therefore, they proposed adding direction information for each dimension of food source position and Eq. 16 is changed as follows:

$$u_{ij} = \begin{cases} x_{ij} + \varphi * (x_{ij} - x_{kj}), & \text{if } d_{ij} = 0 \\ x_{ij} + r * |x_{ij} - x_{kj}|, & \text{if } d_{ij} = 1 \\ x_{ij} - r * |x_{ij} - x_{kj}|, & \text{if } d_{ij} = -1 \end{cases} \quad (18)$$

where u_i is the candidate food source position generated for food source position x_i, d_{ij} is the direction information for j^{th} dimension of the i^{th} food source position and while

φ is a random number in the range of [-1, 1], r is a number generated randomly in the range of [0, 1].

This paper uses the following function to update position of the current food source:

$$v_{ij} = \begin{cases} u_{ij}, & \text{if } r_{ij} < MR \\ x_{ij}, & \text{otherwise} \end{cases} \quad (19)$$

where r_{ij} is a random number generated in the range of [0, 1], u_{ij} is presented in Eq. 18.

Algorithm 2 The pseudo code of the DABC algorithm

Input:
- the maximum cycle number: MCN
- the size of the population: SP
- the number of trials for abandoning food source: $limit$
- the modification rate: MR
- the dimensionality: D

Output: The best individual in the population: $\vec{x}_{best} = \{x_1, x_2, ..., x_D\}$.

Initialize the population solutions x_{ij}, i = 1,...,SP/2, j = 1,...,D.
Compute fitness value for each x_i by using Eq. 15
$cycle$ = 1
while $cycle \leq MCN$ **do**
  for $i = 1$ to $SP/2$ **do**
    - Generate a new solution v_i for the employed bee x_i by using Eq. 19
    - Apply the boundary constraint-handling mechanism for the created solution v_i by using Eq. 20
    - Compute fitness value for each x_i by using Eq. 15
    - Apply the greedy selection process
  end for
  for $i = 1$ to $SP/2$ **do**
    - Compute the probability value p_i for the solution x_i by Eq. 21
  end for
  Formulate the set of potential solutions S by using the roulette-wheel selection mechanism to select $SP/2$ solutions in the population based on the probability value p_i
  for each solution x_i in *S* **do**
    - Generate a new solution v_i for the employed bee x_i by using Eq. 19
    - Apply the boundary constraint-handling mechanism for the created solution v_i by using Eq. 20
    - Compute fitness value for each x_i by using Eq. 15
    - Apply the greedy selection process
  end for
  for $i = 1$ to $SP/2$ **do**
    if value *litmit* of solution x_i is reached **then**
      Produce a random solution and replace x_i with this solution
      break;
    end if
  end for
  Memorize the best solution achieved so far
  $cycle = cycle + 1$
end while

At the beginning of the algorithm, the direction information for all dimensions is equal to 0. If the new solution obtained by Eq. 18 has fitness value better than old one, the direction information will be updated. If prior value of the dimension is less than current value, the direction information of this dimension is set to -1; otherwise it is set to 1. If the fitness of candidate food source is worse than old one, the direction information of the dimension is assigned to 0. This way will help to improve the local search capability and enhance the convergence rate of the algorithm [19].

After generating the candidate food source position, if this position exceeds the boundaries of the variables then boundary constraint-handling mechanism is used and a diverse set of parameter values is produced, which helps to maintain diversity in the population. This paper applies the mechanism proposed in the Kukkonen and Lampinen work [28]. This mechanism is presented as follows:

$$v_{ij} = \begin{cases} 2 * Lb_j - v_{ij}, & \text{if } v_{ij} < Lb_j \\ 2 * Ub_j - v_{ij}, & \text{if } v_{ij} > Ub_j \\ v_{ij}, & \text{otherwise} \end{cases} \quad (20)$$

where v_{ij} is the j^{th} variable of the candidate solution v_i, Lb_j and Ub_j are the lower and upper bounds of the variable v_{ij}. By using Eq. 20, if there are a lot of solutions focused on the extreme values of the search space, the boundary constraint-handling mechanism will help algorithm to avoid getting stuck in the local minimum. After boundary constraint-handling mechanism is applied to the new solution, if the fitness of candidate food source is better than the old one, the new food source position will be memorized and trial counter of the food source is reset; otherwise the trial counter of the food source is increased by 1. This task is called the *greedy selection process*.

In the onlooker phase, each onlooker bee chooses an employed bee based on the probability value associated with food source of the employed bee and improves the quality of the food source chosen by using roulette-wheel selection mechanism [29] with the probability value given as follows:

$$p_i = \frac{fit_i}{\sum_{j=1}^{SP/2} fit_j} \quad (21)$$

where p_i is the being selected probability of the i^{th} employed bee by an onlooker bee. Thereafter, the onlooker bee searches around the food source position chosen and the update process for the current food source position in the onlooker bee phase is the same as in the aforementioned employed bee phase.

In the scout phase, solutions that do not change over a certain number of trials are again initialized by using Eq. 14. In our study, each cycle has a maximum of one scout bee. The details of DABC algorithm are shown in Algorithm 2.

4 Experiments and results

4.1 Experimental dataset

Experiments in this paper have been conducted on a dataset represented by Bailey and Basili [30]. The dataset includes two variables, which are the Kilo Line of code (KLOC) and the Methodology (ME). The measured effort is described in person-months. The dataset is shown in Table 1.

Project No.	KLOC	ME	Measured Effort
1	90.2	30.0	115.8
2	46.2	20.0	96.0
3	46.5	19.0	79.0
4	54.5	20.0	90.8
5	31.1	35.0	39.6
6	67.5	29.0	98.4
7	12.8	26.0	18.9
8	10.5	34.0	10.3
9	21.5	31.0	28.5
10	3.1	26.0	7.0
11	4.2	19.0	9.0
12	7.8	31.0	7.3
13	2.1	28.0	5.0
14	5.0	29.0	8.4
15	78.6	35.0	98.7
16	9.7	27.0	15.6
17	12.5	27.0	23.9
18	100.8	34.0	138.3

Table 1: NASA software project data

In [14] and [15], authors utilized the data of first thirteen projects to optimize parameters of the estimation model, and five remaining projects were used for testing the performance after optimizing the parameters. To ensure comparison with these studies, we also took first thirteen projects as a learning set, and remaining ones are the testing set.

4.2 Experimental setup

The parameters of software effort estimation models are real numbers. With regard to Sheta's Model 1 and Sheta's Model 2, this paper used the range of parameters as presented in [14], and shown in Table 2. Table 3 presents parameter settings for Uysal's Model 1, and Table 4 shows configuration parameters for Uysal's Model 2.

Parameter	Minimum Value	Maximum Value
a	0	10
b	0.3	2
c	-0.5	0.5
d	0	20

Table 2: Configuration parameters for Sheta's Model 1 and Sheta's Model 2

In [14], Sheta used the maximum generation for genetic algorithms to optimize parameters of Sheta's Model 1 and Sheta's Model 2 is 100. This paper, therefore, also used the value of 100 for the maximum cycle number of the DABC algorithm when optimizing parameters of models. The remainder settings of the DABC algorithm to optimize parameters for four software effort estimation models were as follows:

Parameter	Minimum Value	Maximum Value
a	0	10
b	0.3	2
c	-0.5	0.5
d	0	5
e	0	20

Table 3: Parameter settings for Uysal's Model 1

Parameter	Minimum Value	Maximum Value
a	0	10
b	0.3	2
c	-0.5	0.5
d	0	5
e	0	5
f	-5	5
g	0	20

Table 4: Parameter settings for Uysal's Model 2

- The size of the population: 10
- The number of trials for abandoning food source: 2
- The modification rate: 0.7

4.3 Results and empirical evaluations

The model parameters which presented by Eq. 6 were optimized using the DABC algorithm as bellow. This model after optimizing parameters using the DABC was called as **Model 1**.

$$A = 5.4507, \qquad B = 0.7082, \qquad C = -0.3184$$

The model represented by Eq. 7 was named as **Model 2** after its parameters was optimized by using the DABC algorithm. The optimal values of parameters of Model 2 were as bellow:

$$A = 1.7319, \qquad B = 0.966, \qquad C = -0.5, \qquad D = 11.5731$$

This paper calls the software effort estimation model presented by Eq. 8 after optimizing parameters using the DABC as **Model 3**. The optimal values of parameters in Model 3 were as follows:

$$A = 2.3003, \qquad B = 0.8982, \qquad C = -0.0164, \qquad D = 2.0623, \qquad E = 14.1862$$

Model shown by Eq. 9 after optimizing parameters using the DABC was called as **Model 4**. Model 4 got the optimal

values as bellow:

$$A = 4.4442, \quad B = 0.7826, \quad C = -0.373,$$
$$D = 1.2756, \quad E = 2.4425,$$
$$F = -4.9198, \quad G = 17.0804$$

Table 5 shows the actual effort, and predicted effort values using Model 1, Sheta's Model 1, Model 2, Sheta's Model 2, and simple Regression model. Table 6 presents measured data, predicted values of Model 3, Uysal's Model 1, Model 4, Uysal's Model 2, and simple Regression model.

First of all, we assess the accuracy of models using criteria MMRE, MdMRE, and PRED(25). Table 7 shows the obtained results of nine estimation models on 18 NASA projects. The results indicated that Model 1 could not improve the MMRE of Sheta's Model 1, while Model 2 significantly enhanced the MMRE of Sheta's Model 2 by 46.5%. After applying the DABC algorithm, the MMRE of Uysal's Model 1 decreased by more than 5.8% and an additional 5.6% improvement over Uysal's Model 2 was obtained. Model 2, Model 3, and Model 4 produced the better results compared with simple regression with regard to MMRE. This means that the efficiency of software cost estimation models except for Model 1 has been significantly improved after using the DABC algorithm to optimize parameters. With regard to MdMRE, Model 4 gave the lowest result, the second one belonged to Model 3, while Sheta's Model 2 produced the highest value. In general, the values of MdMRE for models with parameters optimized using the DABC are lower than those using original models.

With respect to PRED(25), Model 4, Model 3, and Regression produced the highest value, while the lowest value belonged to Sheta's Model 2. An improvement of 5.5% was achieved in PRED(25) on Uysal's models after applying the DABC algorithm for optimizing parameters. Three out of four improved models gave the values of PRED(25) higher than 75%. This proved that the improved models have been good effort estimation systems. Meanwhile, the values of PRED(25) of both models proposed by Sheta were less than 75%. These results indicated that the models of Sheta have not been suitable for making effort estimates.

In general, the improved models using the DABC algorithm to optimize parameters presented the good prediction accuracy, and were better than the original models in terms of all evaluation criteria. It is also seen that Model 4 gave the most accurate predictions.

As mentioned above, the MMRE criterion is biased, so MAR is used for evaluating the performance of predicted models. The values of MAR of each model on 18 projects are reported in Table 8.

Based on the results of the experiments, it is seen that all improved models outperformed their original ones in terms of MAR. It is also found that the Model 4 overcame all models, while the Model 3 was ranked second. Although Sheta's models were enhanced by using DABC to optimize parameters, they could not show better performance when compared with Uysal's models and the simple regression model. These results continue to prove that Sheta's models are not suitable for software effort estimation.

To enrich the study of the MAR results of the Model 4, we carried out statistical tests to see whether the Model 4 is statistically different from other models in datasets. We also applied statistical tests to see whether the Model 3, the second winning model, is statistically different from other models. We used a normality test on the results obtained and identified that data were not normally distributed. For this reason, we employed the Wilcoxon test, which is a nonparametric test; this type of tests should be used when the distribution is not normal. Table 9 gives the results of the Wilcoxon test based on the 95 % confidence interval (CI). Bold text indicates that the model is statistically different at 95% CI. If the p-value is less than or equal to 0.05, then we conclude that the current model is statistically different from the other models at 95% CI. Otherwise, models are not statistically different at 95% CI. Based on Table 9, we can see that the Model 4 is statistically different from the model 3, and Sheta's Model 2, but it fails to be statistically different from remaining models. Model 3 is statistically different from the Model 2 and Sheta's Model 2, but it is not also statistically different from the remaining models.

5 Conclusion and future work

In this paper, we studied the problem of optimizing parameters for software effort estimation models. The directed artificial bee colony algorithm with several modifications was used to tackle this optimization problem. The models after optimizing parameters produced the results whose accuracy was considerably improved in comparison with the original models in terms of all evaluation criteria such as MMRE, MdMRE, PRED(25), and MAR. In general, the accuracy of software effort is enhanced by more than 5.5% by applying the improved models.

The future work focuses on employing the DABC algorithm to optimize parameters for other effort estimation models. We also use other algorithms of Swarm Intelligence for four models presented in this paper and carry out the comprehensive assessment in terms of the performance of algorithms for the cost estimation problem.

References

[1] Ochodek, M., Nawrocki, J., Kwarciak, K. (2011). Simplifying effort estimation based on Use Case Points, *Information and Software Technology*, 53(3): 200-213.

[2] Mendes, E., Mosley, N., Watson, I. (2002). A comparison of case-based reasoning approaches. *Proceedings of the 11th international conference on World Wide Web*, Hawaii, USA, pp. 272-280.

Project	Measured Effort	Model 1	Sheta's Model 1	Model 2	Sheta's Model 2	Regression
1	115.8	122.617	124.8585	130.6186	134.0202	126.5132
2	96	75.9229	74.8467	71.8103	84.1616	78.6798
3	79	76.6194	75.4852	72.7508	85.0112	80.5612
4	90.8	86.1377	85.4349	83.9645	94.9828	90.4495
5	39.6	51.0323	50.5815	41.9945	56.658	35.4272
6	98.4	98.4043	99.0504	98.378	107.2609	95.7798
7	18.9	24.8788	24.148	18.9008	32.6461	22.5813
8	10.3	17.992	18.0105	11.3608	25.0755	7.6717
9	28.5	38.002	37.2724	29.6205	44.3086	27.6381
10	7	3.8676	4.5849	3.7394	14.4563	8.8264
11	9	9.0108	8.9384	9.0007	19.9759	20.5784
12	7.3	13.4767	13.5926	8.6707	21.5763	8.2111
13	5	0.303	1.51	1.1195	11.2703	4.4963
14	8.4	7.8061	8.2544	5.2715	17.0887	7.1526
15	98.7	108.7481	110.5249	111.4279	118.0378	102.7839
16	15.6	18.648	18.2559	13.6236	26.8312	16.7294
17	23.9	24.0082	23.369	17.9404	31.6864	20.6999
18	138.3	132.1635	135.4825	143.8064	144.4587	135.7203

Table 5: Measured data, predicted values according to Model 1, Sheta's Model 1, Model 2, Sheta's Model 2, and Regression model

Project	Measured Effort	Model 3	Uysal's Model 1	Model 4	Uysal's Model 2	Regression
1	115.8	127.1478	124.794	125.3071	124.3563	126.5132
2	96	78.2194	81.6608	77.743	81.6143	78.6798
3	79	79.4325	83.1941	79.1179	83.1781	80.5612
4	90.8	89.7282	92.8603	89.2478	92.757	90.4495
5	39.6	39.5325	39.0238	39.5424	39.6279	35.4272
6	98.4	98.3011	98.0132	97.2828	97.8566	95.7798
7	18.9	23.3181	23.8838	21.3727	23.8446	22.5813
8	10.3	9.5814	7.7948	8.5967	8.2993	7.6717
9	28.5	30.8556	30.8864	29.6235	31.0829	27.6381
10	7	6.96	5.3694	6.441	5.7918	8.8264
11	9	15.4219	16.4089	14.9203	16.7359	20.5784
12	7.3	9.223	7.6178	7.75	7.8978	8.2111
13	5	2.8414	0.2631	3.3478	0.9986	4.4963
14	8.4	6.9379	5.1496	5.6857	5.4465	7.1526
15	98.7	105.0602	102.5719	104.7675	102.7935	102.7839
16	15.6	17.2108	17.0202	15.2793	17.0407	16.7294
17	23.9	21.7401	21.9803	19.8065	21.9698	20.6999
18	138.3	135.5447	131.2398	133.8053	130.9554	135.7203

Table 6: Measured data, predicted values according to Model 3, Model 4, Uysal's Model 1, Uysal's Model 2, and Regression model

Model	MMRE(%)	MdMRE(%)	PRED(25)
Sheta's Model 1	23.79	14.5	61.11
Sheta's Model 2	63.64	49.27	38.89
Uysal's Model 1	20.04	8.2	77.78
Uysal's Model 2	18.80	8.63	77.78
Regression	17.20	10.31	83.33
Model 1	26.03	14.86	61.11
Model 2	17.13	11.48	77.78
Model 3	14.20	8.65	83.33
Model 4	13.21	7.07	83.33

Table 7: Results for techniques based on criteria MMRE, MdMRE, and PRED(25)

Model	MAR
Sheta's Model 1	5.71
Sheta's Model 2	11.26
Uysal's Model 1	4.00
Uysal's Model 2	3.92
Regression	3.94
Model 1	5.69
Model 2	5.25
Model 3	3.51
Model 4	3.45

Table 8: Results for techniques based on MAR

	p-value at 95% CI
Model 4 vs. Model 3	**0.00072**
Model 4 vs. Model 2	0.61708
Model 4 vs. Model 1	0.18352
Model 4 vs. Uysal's Model 1	0.34722
Model 4 vs. Uysal's Model 2	0.25014
Model 4 vs. Sheta's Model 1	0.20054
Model 4 vs. Sheta's Model 2	**0.0002**
Model 4 vs. Regression	0.28462
Model 3 vs. Model 2	**0.0002**
Model 3 vs. Model 1	0.5892
Model 3 vs. Uysal's Model 1	0.37346
Model 3 vs. Uysal's Model 2	0.5287
Model 3 vs. Sheta's Model 1	0.5892
Model 3 vs. Sheta's Model 2	**0.0002**
Model 3 vs. Regression	0.32708

Table 9: Wilcoxon test for the Model 4 and the Model 3

[3] Jorgensen, M. (2004). A review of studies on expert estimation of software development effort, *Journal of Systems and Software*, 70(1–2): 37-60.

[4] Khalifelu, Z.A., Gharehchopogh, F.S. (2012). Comparison and evaluation of data mining techniques with algorithmic models in software cost estimation, *Procedia Technology*, 1: 65-71.

[5] Briand, L.C., Wieczorek, I. (2002). Resource Estimation in Software Engineering, *Encyclopedia of Software Engineering, John Wiley & Sons*, 2: 1160-1196.

[6] Chen, Z., Menzies, T., Port, D., Boehm, B (2005). Feature subset selection can improve software cost estimation accuracy. Proceedings of the 2005 workshop on Predictor models in software engineering, ACM, pp. 1-6.

[7] Attarzadeh, I., Ow, S.H. (2010). A Novel Algorithmic Cost Estimation Model Based on Soft Computing Technique, *Journal of Computer Science*, 6(2): 117-125.

[8] Putnam, L.H. (1978). A General Empirical Solution to the Macro Software Sizing and Estimating Problem, *IEEE Transactions on Software Engineering*, SE-4(4) 345-361.

[9] Galorath, D.D., Evans, M.W. (2006). *Software sizing, estimation, and risk management*, Auerbach Publications, Boston, MA.

[10] Mittal, A., Parkash, K., Mittal, H. (2010). Software cost estimation using fuzzy logic, *ACM SIGSOFT Software Engineering Notes*, 35(1): 1-7.

[11] Gharehchopogh, F.S. (2011). Neural networks application in software cost estimation: A case study. International Symposium on Innovations in Intelligent Systems and Applications (INISTA), pp. 69-73.

[12] Selby, R.W., Porter, A.A. (1988). Learning from examples: generation and evaluation of decision trees for software resource analysis, *IEEE Transactions on Software Engineering*, 14(12): 1743-1757.

[13] Boehm, B., Abts, C., Chulani, S. (2000). Software development cost estimation approaches — A survey, *Annals of Software Engineering*, 10(1-4): 177-205.

[14] Sheta, A.F. (2006). Estimation of the COCOMO Model Parameters Using Genetic Algorithms for NASA Software Projects, *Journal of Computer Science*, 2(2): 118-123.

[15] Uysal, M. (2010). *Estimation of the Effort Component of the Software Projects Using Heuristic Algorithms*, In RAMOV, B. (ed.). New Trends in Technologies, Rijeka, Croatia, InTech.

[16] Karaboga, D., Basturk, B. (2007). A powerful and efficient algorithm for numerical function optimization: artificial bee colony (ABC) algorithm, *Journal of Global Optimization*, 39(3): 459-471.

[17] Dorigo, M., Maniezzo, V., Colorni, A., Maniezzo, V. (1991). Positive feedback as a search strategy. Technical Report, Politecnico di Milano, Italy.

[18] Kennedy, J., Eberhart, R. (1995). Particle swarm optimization. Proceedings of IEEE International Conference on Neural Networks, pp. 1942-1948.

[19] Kiran, M.S., Findik, O. (2015) A directed artificial bee colony algorithm, *Applied Soft Computing*, 26: 454-462.

[20] Boehm, B., Clark, B., Horowitz, E., Westland, C., Madachy, R., Selby, R. (1995). Cost Models for Future Software Life Cycle Processes: COCOMO 2.0, *Annals of Software Engineering*, 1(1): 57-94.

[21] Boehm, B.W. (1984). Software Engineering Economics, *IEEE Transactions on Software Engineering*, SE-10(1): 4-21.

[22] Olatunji, S., Selamat, A. (2015). Type-2 Fuzzy Logic Based Prediction Model of Object Oriented Software Maintainability, *Intelligent Software Methodologies, Tools and Techniques*, 513: 329-342.

[23] Valdes, F., Abran, A. (2010). Comparing the Estimation Performance of the EPCU Model with the Expert Judgment Estimation Approach Using Data from Industry, *Software Engineering Research, Management and Applications*, 296: 227-240.

[24] Conte, S.D., Dunsmore, H.E., Shen, V.Y. (1986). *Software engineering metrics and models*, Benjamin-Cummings Publishing Co., Inc.

[25] Shepperd, M., MacDonell, S. (2012). Evaluating prediction systems in software project estimation, *Information Software Technology*, 54(8): 820–827

[26] Briand, L., Wieczorek, I. (2002). *Resource Modeling in Software Engineering*. Encyclopedia of Software Engineering, Wiley.

[27] Akay, B., Karaboga, D. (2012). A modified Artificial Bee Colony algorithm for real-parameter optimization, *Information Sciences*, 192: 120-142.

[28] Kukkonen, S., Lampinen, J. (2006). Constrained Real-Parameter Optimization with Generalized Differential Evolution, *IEEE Congress on Evolutionary Computation*, pp. 207-214.

[29] Blickle, T., Thiele, L. (1996). A comparison of selection schemes used in evolutionary algorithms, *Evolutionary Computation*, 4(4): 361-394.

[30] Bailey, J.W., Basili, V.R. (1981). A meta-model for software development resource expenditures. *Proceedings of the 5th international conference on Software engineering*, pp. 107-116.