Software Qual J (2013) 21:501–526
DOI 10.1007/s11219-012-9183-x

A PSO-based model to increase the accuracy of software development effort estimation

Vahid Khatibi Bardsiri · Dayang Norhayati Abang Jawawi · Siti Zaiton Mohd Hashim · Elham Khatibi

Published online: 22 September 2012

Abstract Development effort is one of the most important metrics that must be estimated in order to design the plan of a project. The uncertainty and complexity of software projects make the process of effort estimation difficult and ambiguous. Analogy-based estimation (ABE) is the most common method in this area because it is quite straightforward and practical, relying on comparison between new projects and completed projects to estimate the development effort. Despite many advantages, ABE is unable to produce accurate estimates when the importance level of project features is not the same or the relationship among features is difficult to determine. In such situations, efficient feature weighting can be a solution to improve the performance of ABE. This paper proposes a hybrid estimation model based on a combination of a particle swarm optimization (PSO) algorithm and ABE to increase the accuracy of software development effort estimation. This combination leads to accurate identification of projects that are similar, based on optimizing the performance of the similarity function in ABE. A framework is presented in which the appropriate weights are allocated to project features so that the most accurate estimates are achieved. The suggested model is flexible enough to be used in different datasets including categorical and non-categorical project features. Three real data sets are employed to evaluate the proposed model, and the results are compared with other estimation models. The promising results show that a combination of PSO and ABE could significantly improve the performance of existing estimation models.

Keywords Development effort estimation · Analogy-based estimation · Particle swarm optimization · Software project

V. Khatibi Bardsiri · D. N. A. Jawawi (✉) · S. Z. M. Hashim
Department of Software Engineering, University of Technology Malaysia, Skudai, 81310 Johor Bahru, Malaysia
e-mail: dayang@utm.my

V. Khatibi Bardsiri
e-mail: khatibi78@yahoo.com

E. Khatibi
Department of Computer Engineering, Bardsir Branch, Islamic Azad University, Kerman, Iran

1 Introduction

Time and cost are two important parameters in any type of project, but the importance of these parameters in software projects is much greater than in other projects. There are two main reasons why being on time and on budget are the primary focus in software projects. The first reason is the rapid and dynamic advancements in hardware platforms, which forces software developers to deliver consistent and adaptable software in a limited time frame to fill the gap that exists between software and hardware progress. The second reason is the constant change in customer demands. Nowadays, the diversity of software is very high and the software market has become ever more competitive. Customer requests change every day, and software developers must respond to this event by providing software that is on time and on budget. Moreover, software projects are based on logical and analytical work, while others are based on physical work. We cannot measure the complexity of a software project until we actually begin working on it. Our final object is an intangible product that is very difficult to estimate (Stepanek 2005). Due to the lack of standards for determining the cost and effort of a project, managers and developers are often forced to make decisions regarding these parameters based on prior experience.

The process of improvement in estimation methods has grown steadily; estimation began with very simple assumptions and today it includes complicated equations and techniques. A discussion of existing estimation methods requires dividing the methods into two main groups: algorithmic and nonalgorithmic. The first idea for software development effort estimation based on algorithmic methods was conceived in 1950 by presenting the manual rule of thumb (Jones 2007). As the number of software projects and the need for high-quality software on the part of users increased, new estimation models were presented in 1965 that used linear equations and regression techniques (Boehm and Valerdi 2008). Larry Putnam, Barry Boehm, and Joe Aron were notable pioneers in software estimation methods (Jones 2007). Later, in 1973, IBM researchers presented the first automated tool, Interactive Productivity and Quality (IPQ) (Jones 2007). Barry Boehm proposed a new method called COCOMO that utilized experimental equations to estimate the software development effort (Boehm 1981). In addition, Boehm (1981) explained several algorithms in his book *Software Engineering Economics* that are still used today by researchers. Other models such as Putnam Lifecycle Management (SLIM) and Software Evaluation and Estimation of Resources-Software Estimating Model (SEER-SEM) continued the principles of COCOMO (Boehm and Valerdi 2008).

The introduction of the function point (FP) by Albrecht (Albrecht and Gaffney 1983) was the other important event in that decade. FP was proposed to measure the size of a project in its early stages. All previous estimation methods used Line Of Code (LOC) as the main input parameter, but LOC was a subjective and inaccurate parameter, and most methods suffered from its negative effect on accuracy of estimates. On the contrary, FP was more accurate than LOC in determining the size of a software project based on measuring the functionality of a system (Khatibi Bardsiri and Jawawi 2012). Changes in software development methods and rapid progress in software methodologies led to the development of a new version of COCOMO called COCOMO II in 2000 (Boehm 2000). The new version covered the new features and requirements of software projects. This version of COCOMO included additional details regarding system functionality and utilized FP to determine the scope of projects.

On the other hand, nonalgorithmic estimation methods are constructed based on analyzing completed software projects. Since software projects naturally diff from other types of projects, in many cases algorithmic methods are unable to deal with their dynamic behavior. Moreover, lack of information in the early stages of software projects makes estimating very difficult when algorithmic methods are employed. Therefore, researchers have proposed nonalgorithmic methods to overcome these barriers. Some researchers believed that an estimating process should include expert consultation and opinion. Hence, expert judgment methods were proposed in 1963 (Dalkey and Helmer 1963). In these methods, experts share their ideas in prearranged sessions to achieve agreement on estimation issues. Several studies have tried to analyze the principles of expert judgment methods and to make it easier to use (Molokken and Jorgensen 2005; Jorgensen and Halkjelsvik 2010).

The classification and regression tree (CART) (Breiman, Friedman et al. 1984) was proposed in 1984 as another nonalgorithmic method in this field. Researchers used this method to analyze prior software project data and construct a regression tree in which leaves represented the amount of effort. In such trees, a path is traversed from root to leaf based on the target project features (Briand et al. 1999; Mendes et al. 2003).

The most common non-algorithmic method, analogy-based estimation (ABE), was proposed as a comparative method in 1997 (Shepperd and Schofield 1997). This method predicts the software development effort by comparing the target project with past completed projects and finding projects that most resemble the target project. A comparison process is performed based on project features. Each software project is described through several features (e.g., FP, development type, and application type), which are used by ABE to identify the similarity level of projects.

Due to variations in estimates, some researchers preferred to consider an interval instead of a fixed value for the purpose of estimating (Bakır et al. 2011). Moreover, the effect of data set quality has been investigated as a key point in this area (Bakır et al. 2010).

As neural networks have been widely used for estimation purposes in various sciences, they have also been employed in software development effort estimation (Li et al. 2009a, b; Bhatnagar et al. 2010).

Finally, fuzzy and neuro fuzzy systems have been used in this field to interpret the behavior of software projects from the perspective of effort estimation. Most research studies in this area have tried to model the size of a software project using fuzzy logic to estimate the effort. The COCOMO model has been widely used as the basic model in fuzzy-based research (Ahmed et al. 2005; Azzeh et al. 2010). This paper is organized in six sections as follows. The ABE method and a particle swarm optimization (PSO) algorithm are explained in Sects. 2 and 3, respectively, and the proposed hybrid model is presented in Sect. 4. Section 5 includes the obtained results, and, finally, the conclusion is stated in Sect. 6.

2 Analogy-based estimation (ABE)

The ABE method was proposed by Shepperd in 1997 (Shepperd and Schofield 1997) as a substitute for algorithmic methods. In this method, the estimation of software project metrics is performed by comparing the target project with previously completed projects and finding the projects that are most similar to the target project. Due to its simplicity, ABE has been widely used in software projects many similar predecessors. Basically, ABE consists of four main parts (Khatibi Bardsiri and Jawawi 2011):

(i) Historical data set,
(ii) Similarity function,

(iii) Associated retrieval rules,
(iv) Solution function.

The estimation process of ABE consists of the following steps:

1. Gather data from previous projects and produce a basic data set.
2. Choose proper measurement parameters such as FP and LOC.
3. Retrieve previous projects and calculate the similarities between target project and previous projects.
4. Estimate target project effort.

2.1 Similarity function

ABE uses a similarity function that compares the features of two projects and determines the level of similarity between them. There are two popular similarity functions: Euclidean similarity (ES) and Manhattan similarity (MS) (Shepperd and Schofield 1997). In mathematics, the Euclidean distance or Euclidean metric is the ordinary distance between two points. It is frequently used in optimization problems in which distances only have to be compared. Manhattan distance is a form of geometry in which the usual distance function or metric of Euclidean geometry is replaced by a new metric in which the distance between two points is the sum of the absolute differences of their coordinates.

Both of these functions have been widely used to measure the degree of similarity between software projects. There is no basis for selecting Euclidean or Manhattan functions for a particular data set as this selection is performed by trial and error. The nature of the projects in the data set and the normality level can have a considerable effect on the performance of similarity functions. Equation (1) shows the ES function:

$$\mathrm{Sim}(p,p') = \frac{1}{\left[\sqrt{\sum_{i=1}^{n} w_i \mathrm{Dis}(f_i,f_i')} + \delta\right]} \quad \delta = 0.0001 \tag{1}$$

$$\mathrm{Dis}(f_i,f_i') = \left\{ \begin{array}{ll} (f_i - f_i')^2 & \text{if } f_i \text{ and } f_i' \text{ are numerical or ordinal} \\ 0 & \text{if } f_i \text{ and } f_i' \text{ are nominal and } f_i = f_i' \\ 1 & \text{if } f_i \text{ and } f_i' \text{ are nominal and } f_i \neq f_i' \end{array} \right\}$$

where p and p' are the projects, w_i is the weight, ranging between 0 and 1, assigned to each feature. fi and f_i' display the ith feature of each project, and n indicates the number of features. δ is used to obtain nonzero results. The MS formula is very similar to the ES one, but it computes the absolute difference between features. Equation (2) shows the MS function:

$$\mathrm{Sim}(p,p') = \frac{1}{\left[\sum_{i=1}^{n} w_i \mathrm{Dis}(f_i,f_i') + \delta\right]} \quad \delta = 0.0001 \tag{2}$$

$$\mathrm{Dis}(f_i,f_i') = \left\{ \begin{array}{ll} |f_i - f_i'| & \text{if } f_i \text{ and } f_i' \text{ are numerical or ordinal} \\ 0 & \text{if } f_i \text{ and } f_i' \text{ are nominal and } f_i = f_i' \\ 1 & \text{if } f_i \text{ and } f_i' \text{ are nominal and } f_i \neq f_i' \end{array} \right\}$$

In addition, there are other similarity functions such as rank mean similarity (Walkerden and Jeffery 1997), maximum distance similarity, and Minkowski similarity (Angelis and Stamelos 2000). These three similarity functions have been widely used.

In some previous works, several types of similarity function have been employed to determine the optimum performance of ABE because there is no specific method to indicate the best similarity function to be used in defined conditions (Shepperd and Schofield 1997; Chiu and Huang 2007; Li et al. 2007a, b, 2009a, b; Li and Ruhe 2008a, b).

2.2 Solution function

A solution function is used to estimate the software development effort by considering similar projects that were found to be based on the similarity function. Common solution functions include the most similar project (known as closest analogy) (Walkerden and Jeffery 1999), average of most similar projects (mean) (Shepperd and Schofield 1997), median of most similar projects (median) (Angelis and Stamelos 2000), and inverse distance weighted mean (inverse) (Kadoda et al. 2000). The average describes the mean amount of effort for K most similar projects, where $K > 1$. The median describes the median amount of effort for K most similar projects, where $K > 2$. Finally, the inverse adjusts the portion of each project in estimation using Eq. (3):

$$C_p = \sum_{k=1}^{K} \frac{\mathrm{Sim}(p, p_k)}{\sum_{i=1}^{K} \mathrm{Sim}(p, p_i)} C_{p_k} \tag{3}$$

where p is the new project, p_k the kth most similar project, C_{p_k} the effort value of the kth most similar project p_k, Sim(p, p_k) the similarity between projects p_k and p, and K the total number of most similar projects. Different types of solution functions have been tried; some studies used just one solution function (Huang and Chiu 2006; Li and Ruhe 2008a, b), while other studies considered several types of solution function (Angelis and Stamelos 2000; Li et al. 2009a, b).

2.3 *K* nearest neighborhood (KNN)

Finding the best value of K [in Eq. (3)] has been the subject of several studies in the last decade because the value of K affects the level of accuracy in estimates and may vary in different data sets. The research studies related to analogy methods can be divided based on the method used to find the best number of analogs (K). In a number of studies, the authors considered a fixed number for K (Walkerden and Jeffery 1999; Chiu and Huang 2007). On the other hand, a number of studies determined a range for K and then tried to find the best number (Shepperd and Schofield 1997; Huang and Chiu 2006; Li et al. 2009a, b).

Recently, some researchers have proposed a dynamic search method to find the optimal value of K so that this value may be different for different projects (Li et al. 2007a, b).

2.4 Previous attempts to improve the performance of ABE

Analyzing the correlation coefficient has been employed to improve the performance of ABE. It can be used for feature weighting and feature selection in terms of software development effort estimation. Features that have a weak correlation with effort are given a low weight, and features with a strong correlation are given a high weight. Features with no correlation are deleted. Some studies have demonstrated improvement in the performance of ABE using this technique (Keung et al. 2008; Jianfeng et al. 2009).

Rough set analysis (Pawlak 1991) is a weighting technique that suggests the proper weights for independent features based on a series of predefined criteria. All features are divided into condition features (independent) and decision features (dependent). In this technique, analyzing dependencies between features leads to several reducts (classes), which are subsets of features. Subset discovery is performed using decision rules. The intersection of all classes is considered to comprise core features, which are treated as the most important features. The weighting system in rough set technique is constructed based on the frequency of features in reducts, the existence of features in the core set, and the number of times a feature appears in decision rules. This technique has been used to estimate software development effort (Li et al. 2007a, b; Li and Ruhe 2008a, b).

Gray theory was introduced in 1982 (Deng 1982). Gray indicates a return to the fuzziness concept, black means incomplete and unknown information, and white denotes completed and known information. Gray analysis is a statistical technique used to determine the similarity level between two observations based on a comparison of their features. Use of this technique can improve the performance of the ABE method in terms of comparison (Huang et al. 2008; Azzeh et al. 2010; Hsu and Huang 2011; Song and Shepperd 2011).

One of the most important parts of the ABE method is the solution function because it has a significant effect on the accuracy of predictions. Therefore, several studies have tried to enhance the solution function by applying adjustment expressions (Jorgensen et al. 2003; Chiu and Huang 2007; Li et al. 2009a, b). An adjustment expression refines the results of solution functions to produce more accurate estimations.

Search-based software engineering (SBSE) is a method to apply metaheuristic search techniques like genetic algorithms (GAs), simulated annealing, and tabu search to software engineering problems. Due to the complexity and uncertainty of these problems, common optimization techniques in operations research like linear programming or dynamic programming are mostly impractical for large-scale software engineering problems. This is why researchers and practitioners have utilized metaheuristic search techniques in this area (Harman and Jones 2001).

Despite the fact that the main application of SBSE is in software testing (McMinn 2004), it has been applied to other software engineering activities such as requirements analysis (Greer and Ruhe 2004), software design (Clark and Jacob 2001), software development (Alba and Chicano 2007), and software maintenance (Antoniol et al. 2005). In this paper, SBSE is employed in the field of software development effort estimation by combining ABE and PSO optimization algorithms. In the following sections, studies that have used the SBSE method to increase the accuracy of effort estimation are reviewed.

Optimization techniques can be useful when adjusting the feature weights in the ABE similarity function. GAs, as the most common optimization method, have been used to determine the feature weights in the ABE method (Huang and Chiu 2006; Li et al. 2007a, b, 2008, 2009a, b). Huang and Chiu (2006) defined several linear and nonlinear equations in which the weights are determined for effort drivers. In this study, a GA was employed to search for the best parameters that must be used in the mentioned equations. The results showed that a GA could significantly improve the performance of ABE by adjusting the parameters involved in the equations.

A combination of GA and other techniques has been commonly used to increase the accuracy of effort estimates. Linear adjustment (Chiu and Huang 2007), a gray relational similarity technique (Huang et al. 2008; Hsu and Huang 2011), and regression methods (Oliveira et al. 2010) have been combined with GAs to obtain more accurate results.

Due to the high level of uncertainty, fitness functions or optimization targets play a significant role for effort estimation in software projects. Ferrucci et al. (2010) investigated the efffect of different fitness function on the accuracy of estimates. The results showed that an appropriate selection of performance metrics (which must be optimized) can significantly increase the accuracy of estimates.

The use of a PSO algorithm has been limited to enhancing the performance of the COCOMO model (Lin 2010; Sheta et al. 2010; Reddy 2011). A PSO has no overlapping and mutation calculation. The search can be carried out by the speed of particles. Over several generations, only the most optimistic particle can transmit information to other particles, and the speed of search is very fast. After that the calculation in PSO is very simple. Compared with other calculations under development, PSO offers enhanced optimization and can be completed easily (Bai 2010). Therefore, PSO can be more computationally efficient than GAs in some cases, and it is reasonable to employ this algorithm in field of effort estimation.

3 Particle swarm optimization (PSO) algorithm

Particle swarm optimization (PSO), inspired by the social behavior of birds flocking or fish in schools, is a population-based stochastic optimization technique developed by Kennedy and Eberhart (1995). The main strength of PSO is its fast convergence, which compares with many global optimization algorithms like GAs, simulated annealing, and other global optimization algorithms.

PSO shares many similarities with evolutionary computation techniques such as GAs. The system is initialized with a population of random solutions and searches for optima by generational updating. However, unlike GAs, PSO has no evolution operators such as crossover and mutation. In PSO, the potential solutions, called particles, fly through the problem space by following the current optimum particles. Detailed information will be given in the following sections.

3.1 Optimization process

PSO learns from a scenario and uses it to solve optimization problems. In PSO, each single solution is a "bird" known as a particle in the search space. All particles have fitness values that are evaluated by the fitness function to be optimized and have velocities that direct the flight of the particles. The particles fly through the problem space by following the current optimum particles.

PSO is initialized by a group of random particles (solutions) and then searches for optima by updating generations. In any iteration, each particle is updated by following the two "best" values. The first one is the best solution (fitness) it has achieved so far (the fitness value is also stored). This value is called Pbest. Another "best" value that is tracked by the particle swarm optimizer is the best value obtained so far by any particle in the population. This best value is a global best and called Gbest. When a particle takes part of the population as its topological neighbors, the best value is a local best and is called lbest. After finding the two best values, the particle updates its velocity and position by following Eqs. (4) and (5):

$$v[i] = w * v[i] + c_1 * \text{rand}() * (\text{Pbest}[i] - \text{present}[i]) + c_2 * \text{rand}() * (\text{Gbest} - \text{present}[i]) \quad (4)$$

$$\text{present}[i] = \text{present}[i] + v[i] \quad (5)$$

where v[] is the particle velocity and present[] is the current particle (solution). Pbest[] and Gbest[] are defined as stated previously. rand() is a random number in the interval (0,1), c_1 is a "cognitive parameter," c_2 is a "social parameter," and w is the inertia weight. The combination of these parameters determines the convergence properties of the algorithm.

3.2 Parameter value selection

For the initial version of the PSO, the values for c_1, c_2, and w must be selected. This selection has an effect on the convergence speed and the ability of the algorithm to find the optimum, but different values may be better for different problems. Much work has been done to select a combination of values that works well in a wide range of problems.

The inertia weight determines how the previous velocity of a particle influences the velocity in the next iteration:

- If $w = 0$, then the velocity of the particle is only determined by the part_i and part_g positions; this means that the particle may change its velocity instantly if it is moving far from the best positions recorded in its history. Thus, low inertia weights favor exploitation (local search).
- If w is high, the rate at which a particle may change its velocity is lower (it has an "inertia" that makes it follow its original path) even when better fitness values are known. Thus, high inertia weights favor exploration (global search).

In Shi and Eberhart (1998, 1999), a decaying inertia weight is proposed and tested, with the aim of favoring global search at the start of the algorithm and local search subsequently. If the inertia weight is not reduced with time, the authors suggest selecting a value of $w \in [0.8, 1.2]$. The values of the cognitive and social factors are not critical for the algorithm, but the selection of proper values may result in better performance, both in terms of speed of convergence and alleviation of local minima. Their values must be taken into account when choosing the inertia weight. Several studies propose different values for these parameters that are considered adequate for some of the usual benchmark functions (Table 1).

The pseudocode of the procedure is as follows:

```
For each particle
    Initialize particle
End For

Do
    For each particle
        Calculate fitness value
        If the fitness value is better than the best fitness value (Pbest) in history Then
            Set current value as the new Pbest
        End If
    End For

    Choose the particle with the best fitness value of all the particles as the Gbest
    For each particle
        Calculate particle velocity according to Equation (4)
        Update particle position according to Equation (5)
    End For
```

Table 1 Suggested PSO parameter values

c_1	c_2	w	References
2	2	1	Kennedy et al. (2001)
2	2	0.9	Shi and Eberhart (1999)
1.4962	1.4962	0.7968	Trelea (2003)

Particle velocities in each dimension are clamped to a maximum velocity $V_{\max}$. If the sum of accelerations causes the velocity of that dimension to exceed $V_{\max}$, which is a parameter specified by the user, then the velocity of that dimension is limited to $V_{\max}$.

4 PSO-based estimation model

Even though ABE is a simple, fast, and straightforward estimation method, the nonnormality of software project data sets makes the process of estimation quite difficult. Indeed, the nonnormality of software projects is the main problem that all comparison-based methods like ABE suffer from. As a solution, accurate and efficient feature weighting can considerably improve the performance of ABE. This paper aims to overcome this problem by suggesting appropriate weights for project attributes. To explicitly determine the research goal, three research questions are defined as follows:

1. How can the PSO algorithm be combined with ABE to design an efficient attribute-weighting system?
2. How do the different structures of ABE affect the accuracy of the weighting system?
3. Can a combination of ABE and PSO algorithm increase the accuracy of estimates achieved by other estimation methods?

Since the comparison of a new project with completed projects is the main feature of the ABE method, the accuracy of estimations strongly depends on the integrity of comparisons. Determination of the similarity level that exists between two projects, without regard to the importance of each feature, may have a negative impact on the integrity of comparisons. Due to the uncertain and complex nature of software projects, the process of comparison requires more attention as compared to other projects. Comprehensive feature analysis prior to project comparison can improve the performance of the ABE method. As stated previously, the comparison step is performed through the similarity function in the ABE method. Therefore, the proposed model emphasizes improving the performance of the similarity function. In the proposed hybrid estimation model, the PSO algorithm is combined with the ABE method to increase the accuracy of estimates. Flexibility and adaptability are two valuable specifications of PSO that enable it to overcome the complexity and vagueness of software project features. Indeed, the role of PSO is to find the most appropriate feature weights for use in the similarity function. Weight allocation is performed with regard to the optimization of ABE performance parameters described in the following section. The process of model construction is displayed through a framework including training and testing stages.

4.1 Performance metrics

The performance of estimation methods is evaluated using four main metrics: relative error (RE), magnitude of relative error (MRE), mean magnitude of relative error (MMRE), and percentage of the prediction (PRED), which are computed as follows (Shepperd and Schofield 1997):

$$\mathrm{RE} = \frac{(\mathrm{Estimate} - \mathrm{Actual})}{\mathrm{Actual}} \tag{6}$$

$$\mathrm{MRE} = \frac{|\mathrm{Estimated} - \mathrm{Actual}|}{\mathrm{Actual}} \tag{7}$$

$$\mathrm{MMRE} = \frac{\sum_{i=1}^{N} \mathrm{MRE}}{N} \tag{8}$$

$$\mathrm{PRED}(X) = \frac{A}{N} \tag{9}$$

where A is the number of projects with MRE less than or equal to X, and N is the number of estimated projects. An acceptable level of X in software effort estimation methods is 0.25, and the proposed methods are compared based on this level. MMRE as the total amount of error must be minimized, whereas PRED(0.25) must be maximized.

4.2 Training stage

In the training stage, the proposed estimation model is constructed based on adjusting the feature weights by means of the PSO algorithm to use in the ABE similarity function. The dependent feature (target feature) is development effort; all others are considered independent features. The training stage begins by dividing all available projects into three main groups—basic, training, and testing projects. Projects in the first and second groups are used to construct the estimation model, while testing projects are employed to evaluate the performance of the proposed model. Indeed, the basic projects are the basis of comparison in the proposed model.

In other words, training projects are compared with basic projects to find the most appropriate weights, and testing projects are also compared with the basic projects to examine the accuracy of estimation model. A project is removed from the training group and applied to the similarity function as a new project that needs to be estimated (estimation of project development effort). The PSO algorithm assigns weights (in the range of [0,1]) to the independent features considered in the similarity function (in the first iteration, suggested weights are generated randomly). The removed project is compared with the basic projects by applying PSO suggested weights to Eq. (1). The similarity function discriminates the most similar projects (from basic projects) to the removed project and sends them to the solution function. Eventually, the effort is estimated in a solution function, and the MRE performance metric is computed. This process is repeated until all training projects are estimated.

In the next step, MMRE and PRED(0.25) are computed for a training group based on the obtained amounts of MRE, and these values are passed to the PSO algorithm. Since increasing PRED(0.25) and decreasing MMRE are the main goals of any estimation models in this field, the PSO algorithm is adjusted to minimize the value of (MMRE − PRED(0.25)) as the target parameter. The most efficient weights are found using the PSO algorithm if both MMRE and PRED(0.25) are considered in the optimization process. A low value of MMRE regardless of PRED(0.25) or high value of PRED(0.25) regardless of MMRE cannot be interpreted as an accurate result because there may be biased estimates that change a performance metric to an unreal value. If termination criteria (number

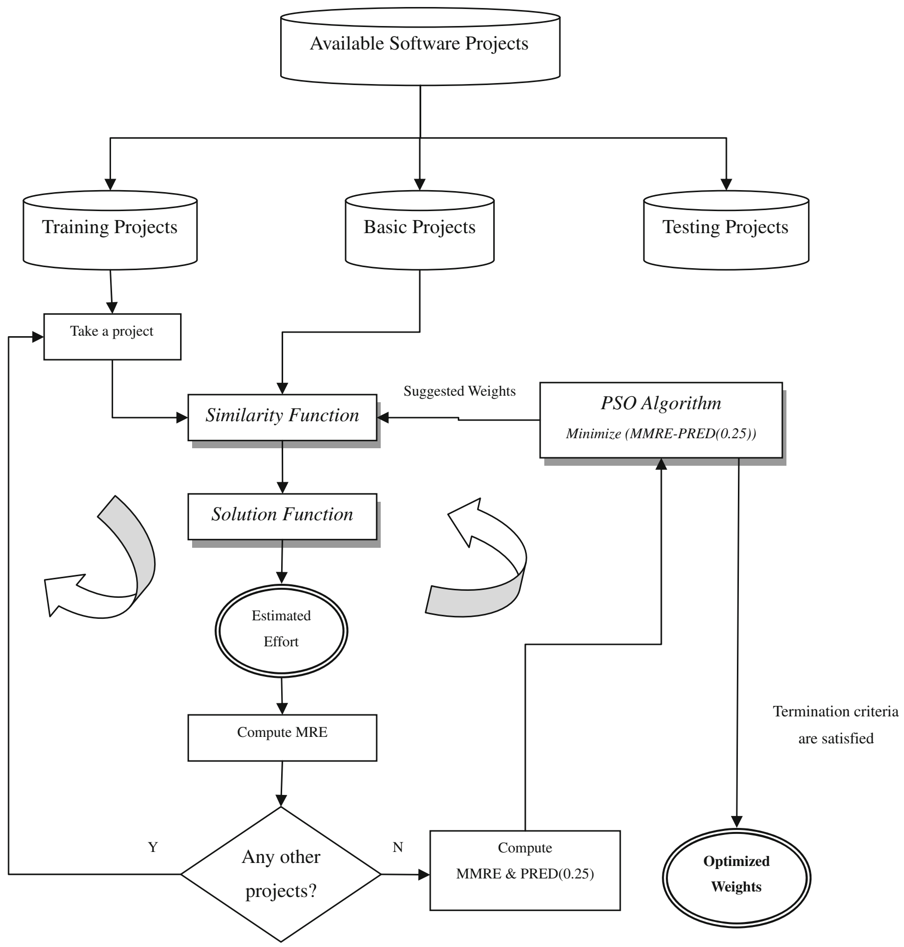

Fig. 1 Training stage of proposed model

of iterations or level of error) are satisfied, the weights are recorded (as optimized weights) to use in the testing stage; otherwise, the PSO algorithm modifies the weights with consideration of the obtained performance parameters. New weights are applied to the similarity function, and all computations are again repeated for the training projects.

This process is continued until termination criteria are satisfied (maximum number of iterations or desired rate of error are achieved). Figure 1 depicts the training stage of the proposed estimation model. As seen in the figure, there are two cycles in the training stage where the first is related to the computation of MMRE for training projects and the second for adjustment of weights by means of the PSO algorithm.

4.3 Testing stage

The main goal of the testing stage is to evaluate the accuracy of the constructed estimation model by applying unseen projects. In this stage, for the purpose of exploring the performance of the proposed model, basic and testing projects are employed as similarity function inputs. In addition, optimized weights, obtained from the training stage, are applied to the similarity function.

Similar to what was done in the training stage, a project is removed from testing projects and then compared with basic projects using the similarity function. Projects that are most similar to the removed project are determined and sent to the solution function. The effort is estimated, and then the amount of MRE is computed. This process is repeated for all testing projects and, finally, performance metrics of MMRE and PRED(0.25) are computed. The testing stage is shown in Fig. 2. As stated in Sect. 4.2, the weights proposed (by PSO) for project features are produced so that the ABE can estimate the training projects as accurately as possible. In this case, basic projects are utilized by ABE for the purpose of comparison. Therefore, two-thirds of the existing projects in the data set (including basic and training sets) are used to reach the best possible weights, and the remaining projects

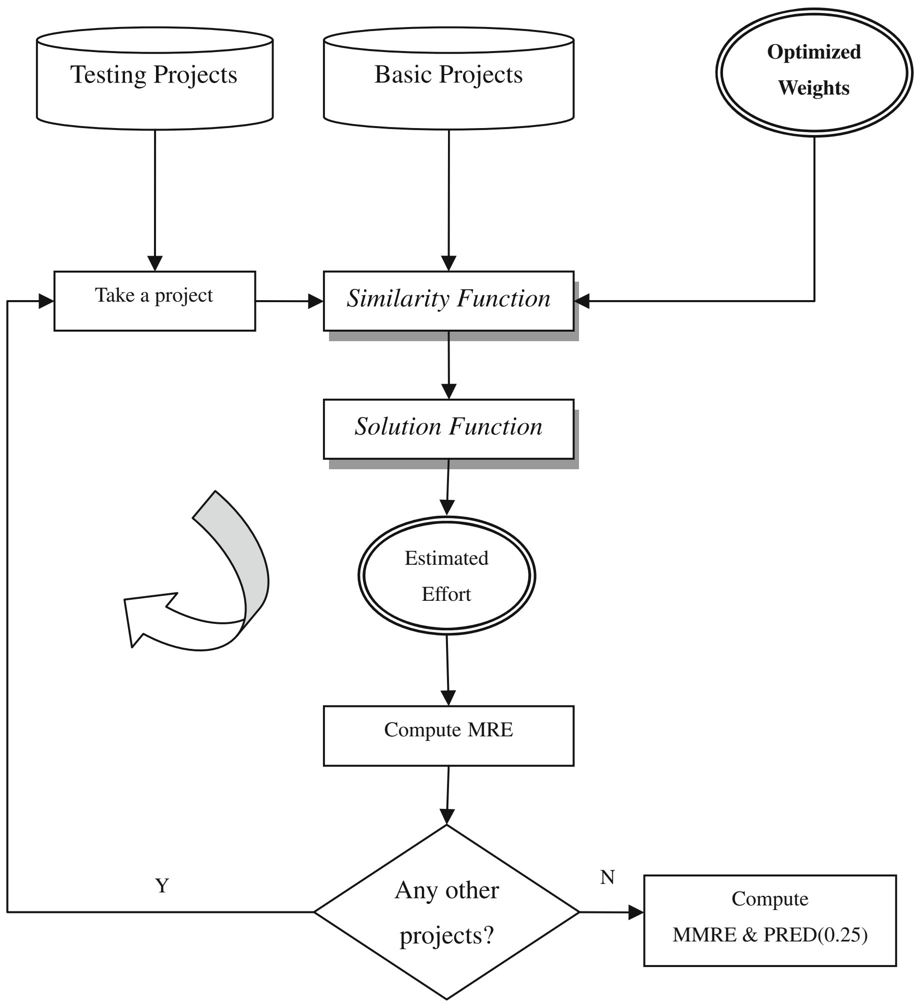

Fig. 2 Testing stage of proposed model

are considered as the test set. PSO randomly generates the weights and adjusts them during the iterations to reach the desired accuracy.

In the proposed model feature weighting is performed so that the combination of weights helps ABE to produce the most accurate results. The value assigned to a feature is a number in the range [0, 1], which is meaningless by itself. PSO is a stochastic algorithm that produces different weights at each iteration, and the best combination of feature weights can be different from one execution to another (different combinations may lead to the same accuracy level). Therefore, the weight assigned to a feature cannot be interpreted as the importance of that feature but as a part of the combination that must be utilized by ABE.

4.4 Evaluation process

If the accuracy of the effort estimation model is computed by applying projects that have been used in the construction of the model, the performance evaluation may be too optimistic. The estimation error may be artificially low and may not reflect the real performance of the model on unseen projects (Hayes 1994). A cross-validation approach gives a more realistic accuracy assessment. It involves dividing the whole data set into multiple training and test sets.

The estimation results in the training stages present the estimation accuracies from the model construction data sets. The testing stages evaluate the estimation accuracies of the models using the other unseen data sets. In the training stage, a cross-validation approach calculates the accuracy for each training data set and then combines the accuracies across all training sets for the training result. In the testing stage, the approach calculates the accuracy for each test set and then combines the accuracies across all test sets for the test result. In this paper, the proposed model is evaluated using threefold cross validation as stated in the following section.

4.4.1 Cross validation

Basic, training, and testing are the three main groups that must be involved in a cross-validation process. Because these three groups are randomly selected from available projects, they can be employed in a k-fold cross validation. Since there are three separate groups of projects, it may seem that threefold cross validation should be considered for the purpose of evaluating, but something is different in this case. In threefold cross validation, all samples are randomly divided into three sets where two are combined to form the training set, and the other is considered as the test set. This classification is performed three times to include all possible situations. Similar to the threefold technique, there are three separate sets in the proposed model but the order of the first two sets (basic and training) considerably affects the model construction process. Therefore, a modification must be applied to the validation process in this case.

Six different sequences can be considered for the proposed model, as seen in Table 2, where S is a set including all projects. S_1, S_2, and S_3 are three subsets selected randomly from S as basic, training, and test sets respectively. These sets include the same number of projects (or approximately the same). At each stage, performance metrics are computed for two separate sequences, and the average is considered as the result of that stage. Final results are determined based on the average of results obtained from all three stages.

Table 2 Applying threefold cross validation to proposed model

S			
S_1		S_2	S_3
	Possible sequences		
	Basic	Training	Testing
Fold 1	S_1	S_2	S_3
	S_1	S_3	S_2
Fold 2	S_2	S_1	S_3
	S_2	S_3	S_1
Fold 3	S_3	S_2	S_1
	S_3	S_1	S_2

4.4.2 Data set description

To explore the real performance, the evaluation of the estimation model must be carried out by applying real data sets. In this study, three data sets are employed to investigate the accuracy of the proposed model. The first data set comes from the IBM data processing services (DPS) organization (Matson et al. 1994) including 24 projects developed by third-generation languages. Five numerical features that may affect the project effort are input count (IC), output count (OC), query count (QC), file count (FC), and adjustment factor (AF). In this data set, there is a project whose effort is more than twice as small as the second smallest project. In practice, this project is not suitable as an analog for other projects. Therefore, it is excluded as an outlier in order to compare the results with previous findings (Chiu and Huang 2007). Table 3 shows the statistical information about this data set.

The second data set, which includes 21 projects, is related to a major Canadian financial (CF) organization (Abran 1996). The collected projects are within the same application domain and developed using a standard development process model. Most of the collected projects developed on the IBM mainframe are included in the data set (Chiu and Huang 2007). IC, OC, inquiry count (IQC), internal logical file (ILF) count, external interface file (EIF), and AF are the main features considered in model construction. Statistical information related to this data set is presented in Table 4.

The third data set is that of the International Software Benchmarking Standards Group (ISBSG 2011), which has been developed and refined by data collection over a 10-year period based on metrics. The latest release of this data set is the ISBSG R11 data repository (ISBSG), which includes a total of 5,052 projects from 24 countries. Statistical information regarding the selected features [Input count (Inpcont), Output count (Outcont), Enquiry count (EnqCont), File count (FileCont), Interface Count (Intcont), Adjusted function point

Table 3 Description of DPS data set

Feature	Minimum	Maximum	Mean	Median	SD
IC	7	193	41.3	34	37.3
OC	12	150	48.7	40	35.3
QC	0	75	17.3	14	19.6
FC	5	60	18	12	15.6
AF	0.75	1.2	1	1	0.1
EFFORT	2.9	105.2	22.8	11.8	28.7

Table 4 Description of CF data set

Variable	Minimum	Maximum	Mean	Median	SD
EIF	2	67	29.9	30	16.2
ILF	0	45	16.6	16	11.3
IC	0	46	17.2	16	11.5
OC	0	69	27.4	25	15.0
IQC	0	33	9.0	8	9.1
AF	66	90	79.8	80	6.2
EFFORT	52	544	331.8	369	151.0

Table 5 Description of ISBSG data set

Variable	Minimum	Maximum	Mean	Median	SD
InpCont	6	1,240	107.7	49.2	172.2
OutCont	4	2,455	165.4	82.5	303.8
EnqCont	3	1,306	98.8	63	137.3
FileCont	7	1,732	114.7	54.5	188.7
IntCont	5	1,572	99.42	36	215.88
AFP	57	7,633	599.1	334.5	864
NorEffort	426	36,225	5,898.9	3,873	6,257.1

(AFP), and Normalized effort in hours (NorEffort)] is displayed in Table 5. Projects with missing values in any of the selected feature are excluded from the subset. In the ISBSG data set, project data quality is rated and only projects with an *A* or *B* rating are used in published research papers. Therefore, projects with ratings other than *A* and *B* are excluded from the subset.

Moreover, since normalized effort (NorEffort) is used as the target for estimation, the risk from using normalized effort should be noted. For projects covering less than a full development life cycle, normalized effort is an estimate of the full development effort, and this may introduce bias. Hence the normalized ratio (normalized effort/summary effort) is used to refine the project subset. As suggested by ISBSG (2011), a ratio of up to 1.2 is acceptable. Projects with a normalized ratio larger than 1.2 are excluded. Finally, the subset is further reduced to the projects with "Insurance" as "OrgType." In the end, the foregoing procedures generated a subset with 134 projects, which are used to evaluate the proposed method.

4.5 Primary adjustments

Data preprocessing is an important part of estimation problems because it can significantly affect quality of training. In this study, all independent features are normalized in a range of [0, 1] to ensure that they have the same effect on the dependent feature (effort). It expedites the process of training in the PSO algorithm. In all data sets, the population size and the number of iterations of the PSO algorithm are adjusted to 100. In addition, to c_1, c_2, and w are assigned 2, 2, and 1, respectively. The mentioned parameters were initialized based on comprehensive trial and error using the PSO algorithm on the training data. MMRE and PRED(0.25) are employed to construct and evaluate the performance of the model because numerous recent studies have used these metrics to compare the accuracy of models (Li et al. 2009a, b; Attarzadeh and Ow 2011; Hsu and Huang 2011;

Table 6 Results on DPS data set

KNN	Testing stage			
	SimFunction	SolFunction	MMRE	PRED(0.25)
$k = 1$	Euclidean	Closest	0.28	0.59
$k = 1$	Manhattan	Closest	0.30	0.56
$k = 2$	Euclidean	Inverse	0.26	0.62
		Mean	0.26	0.61
$k = 2$	Manhattan	Inverse	0.26	0.61
		Mean	0.26	0.60
$k = 3$	Euclidean	Inverse	0.26	0.61
		Mean	0.27	0.60
		Median	0.29	0.55
$k = 3$	Manhattan	Inverse	0.28	0.57
		Mean	0.28	0.58
		Median	0.28	0.54
$k = 4$	Euclidean	Inverse	0.27	0.62
		Mean	0.30	0.57
		Median	0.28	0.58
$k = 4$	Manhattan	Inverse	0.28	0.58
		Mean	0.29	0.56
		Median	0.29	0.56
$k = 5$	Euclidean	Inverse	0.28	0.59
		Mean	0.33	0.55
		Median	0.32	0.55
$k = 5$	Manhattan	Inverse	0.29	0.58
		Mean	0.32	0.56
		Median	0.31	0.54

Khatibi Bardsiri et al. 2011). For the purpose of exploring the role of KNN (described in Sect. 2.3) in terms of the accuracy of the proposed model, five values, $\{1, 2, 3, 4, 5\}$, are applied to the solution function [value of K in Eq. (3) varies from 1 to 5] separately, and the obtained results from each amount of KNN are recorded. Moreover, Euclidean and Manhattan are used as similarity functions to assess whether or not a change in similarity function has any effect on the accuracy of estimates. Three main solution functions, including inverse, mean, and median (Sect. 2.2) are investigated as well. In total, the investigation scope comprises 24 different structures for the proposed model based on a combination of KNN, similarity function, and solution function. The results presented in Tables 6, 7, and 8 were obtained from an iterative execution of PSO (average of 30 executions).

5 Experimental results

5.1 Results on DPS data set

Table 6 shows the results obtained from applying the proposed model to the DPS data set based on the variation of ABE parameters. According to the table, different combinations of model parameters (KNN, similarity function, and solution function) slightly change the

Table 7 Results on CF data set

KNN	Testing stage			
	SimFunction	SolFunction	MMRE	PRED(0.25)
$k = 1$	Euclidean	Closest	0.47	0.53
$k = 1$	Manhattan	Closest	0.47	0.53
$k = 2$	Euclidean	Inverse	0.43	0.61
		Mean	0.41	0.62
$k = 2$	Manhattan	Inverse	0.42	0.60
		Mean	0.41	0.60
$k = 3$	Euclidean	Inverse	0.41	0.63
		Mean	0.41	0.65
		Median	0.43	0.58
$k = 3$	Manhattan	Inverse	0.42	0.63
		Mean	0.40	0.66
		Median	0.43	0.59
$k = 4$	Euclidean	Inverse	0.40	0.65
		Mean	0.39	0.69
		Median	0.41	0.65
$k = 4$	Manhattan	Inverse	0.40	0.65
		Mean	0.39	0.68
		Median	0.41	0.63
$k = 5$	Euclidean	Inverse	0.40	0.68
		Mean	0.40	0.67
		Median	0.41	0.59
$k = 5$	Manhattan	Inverse	0.39	0.68
		Mean	0.38	0.69
		Median	0.40	0.60

accuracy of the model. Regarding the similarity function, since the diversity of results is quite high, it is not possible to determine which similarity function is the best one.

Concerning the solution function, inverse presents more accurate estimations as compared to the others. Moreover, among all values of KNN, $k = 5$ and $k = 2$ produce the worst and best estimates, respectively. In the case of $k = 1$, there is no choice for the solution function and the closest project is selected to estimate the target project. In the case of $k = 2$, mean and median solution functions produce the same results, which is why the median solution function is not shown in the table for this case.

The most accurate estimation (MMRE = 0.26, PRED(0.25) = 0.62) is achieved when KNN, the similarity function, and the solution function are 2, Euclidean, and inverse, respectively. This structure is utilized to compare the proposed model with other estimation models.

5.2 Results on CF data set

The results obtained from applying the proposed estimation model to the CF data set are presented in Table 7. Similar to the DPS data set, there is no significant difference between Euclidean and Manhattan similarity functions in the CF data set. Additionally, the mean

Table 8 Results on ISBSG data set

KNN	Testing stage			
	SimFunction	SolFunction	MMRE	PRED(0.25)
$k = 1$	Euclidean	Closest	0.88	0.31
$k = 1$	Manhattan	Closest	0.85	0.27
$k = 2$	Euclidean	Inverse	0.75	0.46
		Mean	0.79	0.43
$k = 2$	Manhattan	Inverse	0.77	0.45
		Mean	0.78	0.42
$k = 3$	Euclidean	Inverse	0.70	0.46
		Mean	0.76	0.41
		Median	0.79	0.39
$k = 3$	Manhattan	Inverse	0.72	0.45
		Mean	0.74	0.40
		Median	0.80	0.39
$k = 4$	Euclidean	Inverse	0.64	0.51
		Mean	0.69	0.49
		Median	0.72	0.44
$k = 4$	Manhattan	Inverse	0.68	0.49
		Mean	0.70	0.46
		Median	0.71	0.40
$k = 5$	Euclidean	Inverse	0.74	0.45
		Mean	0.77	0.40
		Median	0.82	0.37
$k = 5$	Manhattan	Inverse	0.75	0.43
		Mean	0.77	0.40
		Median	0.80	0.38

solution function produces the best performance metrics in all combinations, whereas median produces the worst results.

Regarding the value of KNN, $k = 1$ presents the worst results, and $k = 4$ and $k = 5$ produce slightly more accurate estimates as compared to the other k values. In general, the best performance of the model [MMRE = 0.38, PRED(0.25) = 0.69] is achieved by a structure of $k = 5$, SimFunction = Manhattan, SolFuntion = mean, which is considered for the purpose of comparing the proposed model with other estimation models.

5.3 Results on ISBSG data set

Table 8 displays the estimates obtained by applying the proposed model to the ISBSG data set. As seen in the table, the similarity function cannot significantly change the accuracy of estimates. The variation in similarity function leads to quite negligible changes in the accuracy of estimates. The level of nonnormality and the diversity of projects in this data set are much greater than in the DPS and CF data sets. Therefore, the range of performance metrics for this data set is quite different from the other ones. Excluding the case of $k = 1$, the inverse solution function produces the most accurate estimates for all other values of KNN. The best performance of the proposed method is achieved by a structure of KNN = 4, SimFunction = Euclidean, SolFunction = inverse, where the value of MMRE is 0.64 and that of PRED(0.25) is 0.51. This structure is considered for the purpose of comparing the proposed method with other estimation methods.

5.4 Proposed model versus other estimation models

The performance of the proposed model is compared with eight well-known estimation models. Since the primary goal of this paper is to enhance the accuracy of the ABE method, performance metrics obtained from the proposed model must be compared with ABE. Therefore, all possible structures of ABE (similar to structures considered for the proposed model) are applied to all data sets and the best performance metrics are recorded for comparison purposes. Moreover, regression-based estimation methods including CART, stepwise regression (SWR), and multiple regression (MLR) are involved in the comparison process because they have been widely used in other research studies (Li et al. 2009a, b; Mittas and Angelis 2010; Hsu and Huang 2011). Finally, artificial neural network (ANN) and three adjusted ABEs (Chiu and Huang 2007)—AAE, AAMH, and AAMK—are compared with the proposed method. All results presented in the following sections are computed using a threefold cross-validation technique.

Table 9 summarizes the results obtained from applying the selected estimation models to the DPS data set (testing data) based on MMRE and PRED(0.25). It can be seen that the proposed model produces the most accurate estimates (MMRE = 0.26, PRED(0.25) = 0.62) in the testing stage as compared to the other methods, followed by AAE [MMRE = 0.38, PRED(0.25) = 0.57]. The worst estimates come from SWR [MMRE = 0.93, PRED(0.25) = 0.16].

The results of applying different estimation models to the CF data set in the testing stage are shown in Table 10. As seen in the table, the proposed model achieves the best performance metrics [MMRE = 0.38, PRED(0.25) = 0.69], among all other estimation models, followed by AAE [MMRE = 0.52, PRED(0.25) = 0.43]. In addition, SWR produces the worst estimates [MMRE = 0.97, PRED(0.25) = 0.02].

Table 11 shows the performance metrics obtained from applying the proposed method to the ISBSG data set. The range of performance metrics for this data set shows its complexity and high nonnormality. As seen in the table, the most accurate estimates are achieved by the proposed estimation model [MMRE = 0.64, PRED(0.25) = 0.51]. The method closest to the proposed model is AAMK [MMRE = 0.89, PRED(0.25) = 0.27]. Moreover, MLR presents the worst estimates on this data set [MMRE = 1.32, PRED(0.25) = 0.16].

Table 9 Comparison of estimation models on DPS data set

Method	MMRE	PRED(0.25)
SWR	0.93	0.16
ANN	0.90	0.22
MLR	0.73	0.30
CART	0.62	0.26
ABE	0.51	0.42
AAMK	0.43	0.61
AAE	0.38	0.57
AAMH	0.36	0.52
PROPOSED	0.26	0.62

Table 10 Comparison of estimation models on CF data set

Method	MMRE	PRED(0.25)
SWR	0.97	0.02
CART	0.99	0.24
MLR	0.98	0.27
ABE	0.84	0.29
ANN	0.70	0.10
AAE	0.52	0.43
AAMH	0.49	0.38
AAMK	0.49	0.38
PROPOSED	0.38	0.69

Table 11 Comparison of estimation models on ISBSG data set

Method	MMRE	PRED(0.25)
MLR	1.32	0.16
ABE	1.21	0.19
SWR	1.13	0.13
ANN	0.96	0.22
CART	0.99	0.18
AAE	0.95	0.28
AAMH	0.91	0.18
AAMK	0.89	0.27
PROPOSED	0.64	0.51

5.5 Improvement analysis

Since improving the accuracy of ABE was the main idea behind this paper, analysis of obtained results must be performed comprehensively to clarify how the proposed model improves the performance of ABE. In this section, percentage of improvement in terms of ABE, AAE, AAMH, and AAMK is investigated (Figs. 3, 4).

Figure 3 depicts the percentage of improvement achieved by applying the proposed model to the DPS data set. It is observed that the proposed model improves the accuracy of ABE in both performance metrics of MMRE and PRED(0.25) by 49 and 48 %, respectively. Since both performance metrics were improved in the same range, it is confirmed that the improvement domain involves a wide range of testing projects. Increasing the accuracy by almost 50 % implies that applying PSO can considerably enhance the performance of ABE on the DPS data set.

The performance of AAE, AAMH, and AAMK was improved as well, but the percentage of improvement is not as good as that of ABE. In terms of AAE and AAMK, improvement of MMRE is quite a bit more than PRED(0.25). As seen in Fig. 3, in these models, PRED(0.25) was improved by less than 10 %. This shows that the range of improvement is limited to a few projects but the percentage of improvement in these few projects is convincing (32 and 40 %).

Regarding AAMH, MMRE and PRED(0.25) were improved by almost the same percentage. Compared to AAE and AAMK, the estimation model of AAMH has a lower improvement percentage of MMRE (28 %) and a higher improvement percentage of PRED(0.25) (19 %). In other words, the percentage of accuracy improvement in AAMH is less than the other two models, but the number of projects affected by this improvement is

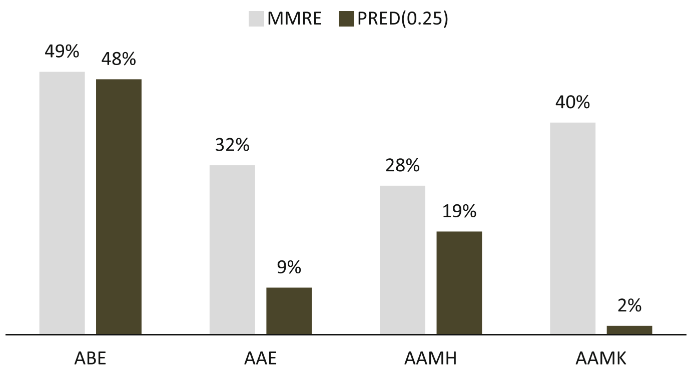

Fig. 3 Percentage of improvement obtained by proposed method on DPS

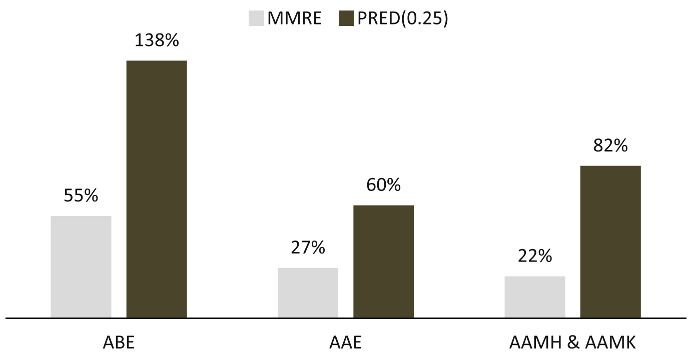

Fig. 4 Percentage of improvement obtained by proposed method on CF

more than the number of improved projects in the other two models. It can be concluded that the enhancement of MMRE is greater than PRED(0.25) in all of the mentioned estimation models. The percentage of improvement obtained from using the proposed model in CF data set is shown in Fig. 4. Since the performance metric values for AAMH and AAMK are the same (Table 10), these models are combined in Fig. 4. According to the figure, the greatest enhancement is achieved for ABE where MMRE and PRED(0.25) are improved by 55 and 138 %, respectively, which indicates that estimation accuracy was increased in a wide range of projects by a convincing percentage. Therefore, PSO can significantly improve the performance of ABE in the CF data set.

MMRE and PRED(0.25) were improved by almost the same percentage as for AAE. Although the improvement of PRED(0.25) in this case is a bit less than that observed in ABE, performance enhancement is still more than 50 %, which certifies the superiority of the proposed model.

In terms of the last two models (AAMH and AAMK), PRED(0.25) is improved by a promising percentage of 82 %, which indicates that a high number of estimates are improved but the percentage of improvement in total accuracy (MMRE = 27 %) is less than in the other two models. The improvement percentage of PRED(0.25) is higher than MMRE in the CF data set. In addition, the proposed model presents a greater improvement percentage in the CF data set compared with the DPS data set.

Figure 5 depicts the percentage of improvement achieved by applying the proposed model to the ISBSG data set. It is observed that the value of PRED(0.25) is increased

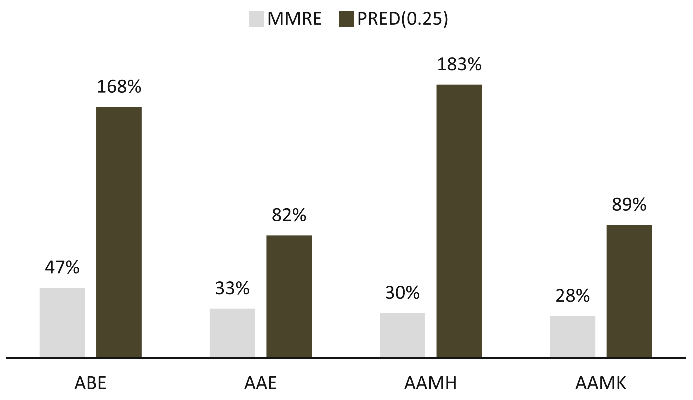

Fig. 5 Percentage of improvement obtained by proposed method on ISBSG

significantly for all types of ABE (improvement of 168, 82, 183, 89 %). The largest improvement of MMRE, 47 %, occurred for ABE and the lowest percentage of improvement, 28 %, is related to AAMK. Compared to the other data sets, the improvement percentage of PRED(0.25) is higher than the others, whereas the improvement percentage of MMRE is lower. This indicates that the number of outliers and nonnormal projects in ISBSG are more than in the other data sets.

Regarding the research questions, the first question was addressed by the explanation of the hybrid model in Sects. 4.2 and 4.3. The training and testing stages showed how the PSO algorithm could be combined with ABE. In addition, the comparison between the proposed hybrid model and ABE (Sect. 5.4) certified the efficiency of the weighting system, as mentioned in the first research question. The second question was investigated in Sects. 5.1–5.3, where the different structures of ABE were employed in the proposed hybrid model. The results proved that the proposed weighting system is flexible enough to be used by different structures of ABE. Finally, the third research question was addressed in Sect. 5.4, where the performance of the proposed model was evaluated against the other estimation models. This evaluation certified that the proposed model outperformed the other models on all three data sets. Therefore, it can be concluded that the combination of ABE and the PSO algorithm can improve the performance of other estimation models.

6 Conclusions

It is undeniable that development effort estimation plays a vital role in software project management. Due to the complexity and inconsistency of software projects, inaccurate estimates have become a common challenging issue, which bothers the developers and managers throughout the development phases. The comparison between new projects and previously completed projects is the main idea behind much research in this field. Although ABE is the best-known comparison-based estimation model and has been widely used in software development effort estimation, it is still unable to produce accurate estimates in many situations. PSO as a low computational cost and fast optimization algorithm was combined with ABE using the framework proposed in this paper. The proposed framework consists of training and testing stages in which an estimation model is constructed and evaluated. PSO explores the possible weights and selects those that will lead to the most accurate estimates. Indeed, the quality of the comparison process in the ABE method was improved by assigning the most appropriate weights to project features. To evaluate the performance of the proposed model, three real data sets were employed, and the performance metrics of MMRE and PRED(0.25) were computed using a cross-validation technique. The encouraging results showed that the proposed model can significantly increase the accuracy of estimates based on MMRE and PRED(0.25). The obtained results were compared with eight common estimation models, which showed the superiority of the proposed model in all data sets. According to the results obtained from three real data sets, it can be concluded that the combination of PSO and ABE leads to a high-performance model in terms of software development effort estimation. Besides achieving promising estimates, the proposed model can be used in a wide range of software project data sets by modifying a few parameters (primary adjustments). There is no prior assumption or prerequisite to use the proposed model. This means that it is a consistent, flexible, and adaptable estimation model that is suitable for use in various types of software projects. As future work, we will use other new optimization algorithms to increase the accuracy of development effort estimation in the ABE method.

Acknowledgments We would like to thank the International Software Benchmarking Standards Group (ISBSG) to approve for accessing the ISBSG data set for our research.

References

Abran, A. (1996). Function points analysis: An empirical study of its measurement processes. *IEEE Transactions on Software Engineering, 22*(12), 895–910.

Ahmed, M. A., Omolade Saliu, M., et al. (2005). Adaptive fuzzy logic-based framework for software development effort prediction. *Information and Software Technology, 47*(1), 31–48.

Alba, E., & Chicano, J. F. (2007). Software project management with GAs. *Information Science, 177*(11), 2380–2401.

Albrecht, A. J., & Gaffney, J. A. (1983). Software function, source lines of codes, and development effort prediction: a software science validation. *IEEE Transactions on Software Engineering, 9*(6), 639–648.

Angelis, L., & Stamelos, I. (2000). A simulation tool for efficient analogy based cost estimation. *Empirical Software Engineering, 5*(1), 35–68.

Antoniol, G., Penta, M. D., et al. (2005). Search-based techniques applied to optimization of project planning for a massive maintenance project. In *Proceedings of the 21st IEEE international conference on software maintenance* (pp. 240–249). IEEE Computer Society.

Attarzadeh, I., & Ow, S. H. (2011). Software development cost and time forecasting using a high performance artificial neural network model. *Intelligent Computing and Information Science, 134*, 18–26.

Azzeh, M., Neagu, D., et al. (2010). Fuzzy grey relational analysis for software effort estimation. *Empirical Software Engineering, 15*(1), 60–90.

Bai, Q. (2010). Analysis of particle swarm optimization algorithm. *Computer and Information Science, 3*(1), 180–184.

Bakır, A., Turhan, B., et al. (2010). A new perspective on data homogeneity in software cost estimation: A study in embedded systems domain. *Software Quality Journal, 18*(1), 57–80.

Bakır, A., Turhan, B., et al. (2011). A comparative study for estimating software development effort intervals. *Software Quality Journal, 19*(3), 537–552.

Bhatnagar, R., Bhattacharjee, V., et al. (2010). Software development effort estimation neural network vs. regression modeling approach. *International Journal of Engineering Science and Technology, 2*(7), 2950–2956.

Boehm, B. W. (1981). *Software engineering economics*. Englewood Cliffs, NJ: Prentice Hall.

Boehm, B. (2000). *Software cost estimation with COCOMO II*. Englewood Cliffs, NJ: Prentice Hall.

Boehm, B. W., & Valerdi, R. (2008). Achievements and challenges in cocomo-based software resource estimation. *IEEE Software, 25*(5), 74–83.

Breiman, L., Friedman, J. H., et al. (1984). *Classification and regression trees*. Pacific Grove, CA: Wadsworth.

Briand, L. C., El-Emam, K., et al. (1999). An assessment and comparison of common cost software project estimation methods. In *Proceedings of 21st international conference software engineering* (pp. 313–322).

Chiu, N. H., & Huang, S. J. (2007). The adjusted analogy-based software effort estimation based on similarity distances. *Journal of Systems and Software, 80*, 628–640.

Clark, J. A., & Jacob, J. L. (2001). Protocols are programs too: The meta-heuristic search for security protocols. *Information and Software Technology, 43*(14), 891–904.

Dalkey, N., & Helmer, O. (1963). An experimental application of the Delphi method to the use of experts. *Management Science, 9*(3), 458–467.

Deng, J. L. (1982). Control problems of grey systems. *Systems and Control Letters, 1*(5), 288–294.

Ferrucci, F., Gravino, C., et al. (2010). Genetic programming for effort estimation: An analysis of the impact of different fitness functions. In *Proceedings of the 2nd international symposium on search based software engineering (SSBSE'10), Benevento, Italy* (pp. 89–98). IEEE.

Greer, D., & Ruhe, G. (2004). Software release planning: an evolutionary and iterative approach. *Information and Software Technology, 46*(4), 243–253.

Harman, M., & Jones, B. F. (2001). Search-based software engineering. *Information and Software Technology, 43*(14), 833–839.

Hayes, W. (1994). *Statistics* (5th ed.). Chicago: Harcourt Brace.

Hsu, C.-J., & Huang, C.-Y. (2011). Comparison of weighted grey relational analysis for software effort estimation. *Software Quality Journal, 19*(1), 165–200.

Huang, S.-J., & Chiu, N.-H. (2006). Optimization of analogy weights by genetic algorithm for software effort estimation. *Information and Software Technology, 48*, 1034–1045.

Huang, S.-J., Chiu, N.-H., et al. (2008). Integration of the grey relational analysis with genetic algorithm for software effort estimation. *European Journal of Operational Research, 188*(3), 898–909.

ISBSG. (2011). International Software Benchmarking standard Group from www.isbsg.org.

Jianfeng, W., Shixian, L., et al. (2009). Improve analogy-based software effort estimation using principal components analysis and correlation weighting. In *International conference on software engineering*.

Jones, C. (2007). *Estimating software costs: Bringing realism to estimating*. New York: McGraw-Hill.

Jorgensen, M., & Halkjelsvik, T. (2010). The effects of request formats on judgment-based effort estimation. *Journal of Systems and Software, 83*(1), 29–36.

Jorgensen, M., Indahl, U., et al. (2003). Software effort estimation by analogy and regression toward the mean. *Journal of Systems and Software, 68*, 253–262.

Kadoda, G., Cartwright, M., et al. (2000). *Experiences using case-based reasoning to predict software project effort international conference on empirical assessment and evaluation in software engineering*. KEELE University.

Kennedy, J., & Eberhart, R. (1995). Particle swarm optimization. In *IEEE international conference on neural networks, Piscataway, NJ*.

Kennedy, J., Eberhart, R. C., et al. (2001). *Swarm intelligence*. San Francisco: Morgan Kaufmann.

Keung, J. W., Kitchenham, B. A., et al. (2008). Analogy-X: Providing statistical inference to analogy-based software cost estimation. *IEEE Transactions on Software Engineering, 34*(4), 471–484.

Khatibi Bardsiri, V., & Jawawi, D. N. A. (2011). Software cost estimation methods: A review. *Journal of Emerging Trends in Computing and Information Sciences, 2*(1), 21–29.

Khatibi Bardsiri, V., & Jawawi, D. N. A. (2012). *Software development effort estimation*. Germany: Lambert Academic Publishing.

Khatibi Bardsiri, V., Jawawi, D. N. A., et al. (2011). A new fuzzy clustering based method to increase the accuracy of software development effort estimation. *World Applied Sciences Journal, 14*(9), 1265–1275.

Li, J., & Ruhe, G. (2008a). Analysis of attribute weighting heuristics for analogy-based software effort estimation method AQUA+. *Empirical Software Engineering, 13*(1), 63–96.

Li, J. Z., & Ruhe, G. (2008b). Software effort estimation by analogy using attribute selection based on rough set analysis. *International Journal of Software Engineering and Knowledge Engineering, 18*(1), 1–23.

Li, J., Ruhe, G., et al. (2007). A flexible method for software effort estimation by analogy. *Empirical Software Engineering, 12*(1), 65–106.

Li, Y. F., Xie, M., et al. (2007). A study of genetic algorithm for project selection for analogy based software cost estimation. In *International conference on industrial engineering and engineering management, Singapore*.

Li, Y. F., Xie, M., et al. (2008). A study of analogy based sampling for interval based cost estimation for software project management. In *4th IEEE international conference on management of innovation and technology, Singapore*.

Li, Y. F., Xie, M., et al. (2009a). A study of project selection and feature weighting for analogy based software cost estimation. *Journal of Systems and Software, 82*(2), 241–252.

Li, Y. F., Xie, M., et al. (2009b). A study of the non-linear adjustment for analogy based software cost estimation. *Empirical Software Engineering, 14*, 603–643.

Lin, J.-C. (2010). Applying particle swarm optimization to estimate software effort by multiple factors software project clustering. In *International computer symposium (ICS), Taiwan*.

Matson, J. E., Barrett, B. E., et al. (1994). Software development cost estimation using function points. *IEEE Transactions on Software Engineering, 20*(4), 275–287.

McMinn, P. (2004). Search-based software test data generation: A survey: Research articles. *Software Testing, Verification, and Reliability, 14*(2), 105–156.

Mendes, E., Watson, I., et al. (2003). A comparative study of cost estimation models for web hypermedia applications. *Empirical Software Engineering, 8*, 163–196.

Mittas, N., & Angelis, L. (2010). LSEbA: Least squares regression and estimation by analogy in a semi-parametric model for software cost estimation. *Empirical Software Engineering, 15*(5), 523–555.

Molokken, K., & Jorgensen, M. (2005). Expert estimation of Web-development projects: Are software professionals in technical roles more optimistic than those in non-technical roles? *Empirical Software Engineering, 10*(1), 7–30.

Oliveira, A. L. I., Braga, P. L., et al. (2010). GA-based method for feature selection and parameters optimization for machine learning regression applied to software effort estimation. *Information and Software Technology, 52*(11), 1155–1166.

Pawlak, Z. (1991). *Rough set: Theoretical aspects of reasoning about data*. Dordrecht: Kluwer.

Reddy, P. (2011). Particle swarm optimization in the fine-tuning of fuzzy software cost estimation models. *International Journal of Software Engineering and Knowledge Engineering, 1*(2), 12–23.

Shepperd, M., & Schofield, C. (1997). Estimating software project effort using analogies. *IEEE Transactions on Software Engineering, 23*(11), 736–743.

Sheta, A. F., Ayesh, A., et al. (2010). Evaluating software cost estimation models using particle swarm optimisation and fuzzy logic for NASA projects; a comparative study. *International Journal of Bio-Inspired Computation, 2*(6), 365–373.

Shi, Y., & Eberhart, R. C. (1998). Parameter selection in particle swarm optimization. In *The 7th annual conference on evolutionary programming*.

Shi, Y., & Eberhart, R. C. (1999). Empirical study of particle swarm optimization. In *IEEE congress on evolutionary computation*.

Song, Q., & Shepperd, M. (2011). Predicting software project effort: A grey relational analysis based method. *Expert Systems with Applications, 38*(6), 7302–7316.

Stepanek, G. (2005). *Software project secrets: Why software projects fail*. Berkeley, CA: Apress.

Trelea, I. C. (2003). The particle swarm optimization algorithm: Convergence analysis and parameter selection. *Information Processing Letters, 85*(6), 317–323.

Walkerden, F., & Jeffery, R. (1997). Software cost estimation: A review of models, process, and practice. *Advances in Computers, 44*, 59–125.

Walkerden, F., & Jeffery, R. (1999). An empirical study of analogy-based software effort estimation. *Empirical Software Engineering, 4*(2), 135–158.

Author Biographies

Vahid Khatibi Bardsiri is a lecturer at Islamic Azad University, Bardsir Branch, Iran. He holds B.Sc. and M.Sc. degrees in software engineering from Ferdowsi University of Mashhad (2002) and Science and Research Branch of Islamic Azad University (2004). Currently, he is working on his Ph.D. in Software Metrics Estimation at Universiti Teknologi Malaysia (UTM). His research interests include agile software development methods, artificial intelligence, and software measurement.

Dayang Norhayati Abang Jawawi is a lecturer in the Department of Software Engineering, Faculty of Computer Science and Information Systems, Universiti Teknologi Malaysia (UTM). She received her B.Sc. in Software Engineering from Sheffield Hallam University, UK, and conducted her M.Sc. and Ph.D. research in Software Engineering from Universiti Teknologi Malaysia. Her research interests are in software reuse, software quality, and software testing. Currently she is Head of the Software Engineering Department, Faculty of Computer Science and Information Systems, UTM, and a member of the Software Engineering Research Group (SERG), K-Economy, UTM. Most of her research projects focus on rehabilitation and mobile robotics, real-time embedded systems, and precision farming applications.

Siti Zaiton Mohd Hashim is an Associate Professor at the Department of Software Engineering, Faculty of Computer Science and Information Systems, Universiti Teknologi Malaysia (UTM). She received her B.Sc. Degree in Computer Science from Univ of Harford, USA, M.Sc. in Computing from University of Bradford, UK and Ph.D. research in Soft Computing from University of Sheffield, UK. Her research interests are Soft Computing techniques and applications System Development and Intelligent System. Currently she is Deputy Dean, Office of Postgraduate, Faculty of Computer Science and Information Systems, UTM and a member of Soft Computing Research Group (SCRG), K-Economy, UTM.

Elham Khatibi received her B.Sc. Degree in Mathematics from Islamic Azad University, Kerman, Iran (2007). Currently, she is undertaking her M.Sc. in Information Technology Management at Universiti Teknologi Malaysia (UTM). Her research interests are software project management and software development methodologies.