Utilizing Faults and Time to Finish Estimating the Number of Software Test Workers Using Artificial Neural Networks and Genetic Programming

Alaa Sheta[1](✉), Sultan Aljahdali[2], and Malik Braik[3]

[1] Department of Computing Sciences, Texas A&M University-Corpus Christi, Texas, TX 78412, USA
alaa.sheta@tamucc.edu

[2] Computer Science Department, College of Computers and Information Technology, Taif, Saudi Arabia
aljahdali@tu.edu.sa

[3] Department of Computer Science, Al-Balqa Applied University, Salt, Jordan
mbraik@bau.edu.jo

Abstract. Time, effort and the estimation of number of staff desired are critical tasks for project managers and particularly for software projects. The software testing process signifies about 40–50% of the software development lifecycle. Faults are detected and corrected during software testing. Accurate prediction of the number of test workers necessary to test a software before the delivery to a customer will save time and effort. In this paper, we present two models for estimating the number of test workers required for software testing using Artificial Neural Networks (ANN) and Genetic Programming (GP). We utilize the expected time to finish testing and the rate of change of fault observation as inputs to the proposed models. The proposed models were able to predict the required team size; thus, supporting project managers in allocating the team effort to various project phases. Both models yielded promising estimation results in real-time applications.

Keywords: Prediction of test workers · Software testing · Project management Artificial Neural Networks · Genetic Programming

1 Introduction

Software testing process is defined as the process of implementing a program with the intent to find software bugs, errors or any defects [1]. It requires numerous efforts and might cost more than 50% of the project development effort [1]. This process should deliver software with minimum or no faults. The software development life cycle is all about people, methodologies and tools. This is evident from the software development process (see Fig. 1). People (i.e., staff) need to collect the project requirements, develop a project plan, make a design, deploy the project, test and validate the business requirements and finally detect and fix the bugs if any. The standard software development life cycle consists of multiple stages: requirements, analysis and design, coding, unit and system test and finally software evolution. The staff management process

https://doi.org/10.1007/978-3-030-03577-8_67

for the project development involves the following five phases: Staff Planning, Staff Acquisition, Staff Training, Staff Tracking, and Staff Transition. This process appears in Fig. 2. Specific information related to staff must be collected, organized and updated during the project development life cycle. The project manager should be able to identify the size of the team required to test the software that primarily relies on the expected number of faults. Employing a large team with no need means money loss; while employing a small team with a lot of bugs in the software means a delay on the delivery day. Therefore, a compromise must be reached. This can be achieved by building a model that can estimate the required number of test workers to utilize the bugs or faults in the software.

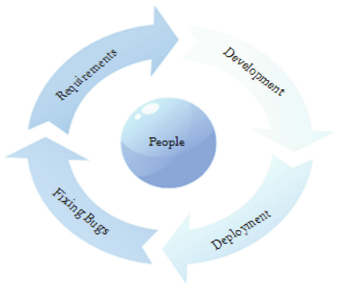

Fig. 1. Software development process [4].

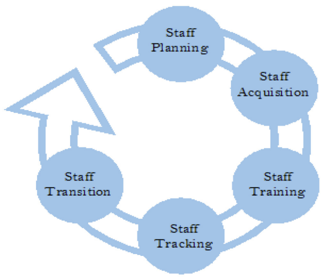

Fig. 2. Staff management process [4].

More recently, there has been a growing interest in the use of evolutionary computation and soft computing techniques such as Genetic Algorithms (GAs), Genetic Programming (GP), Artificial Neural Networks (ANNs), Fuzzy Logic (FL), Particle Swarm Optimization (PSO) and Grey Wolf Optimizer (GWO) to solve a variety of software engineering problems such as estimating software effort and detecting faults during software testing [2–5]. Sheta [2] presented two models using GAs to estimate the effort required to develop software projects. Several models in predicting the software cost using PSO, FL and other cost estimation models were presented in [3]. Sheta et al. [4] showed an estimate of the number of test workers necessary for a software testing process using ANN. In [5], the authors used GWO to estimate the parameters of the software reliability growth model in order to reduce the difference between the expected and the actual number of failures of the software system.

In this paper, we use ANN and GP approaches to developing models for estimating the number of test workers required to test software utilizing the number of measured faults. Experiments are conducted for two different projects to evaluate the performance of the developed models. This paper is organized as follows. In Sect. 2, an overview of the artificial neural network is given with an insight into the multilayer Perceptron architecture and the learning algorithm used. Section 3 presents the basic concepts of genetic programming. The test data used for training and testing the models is presented in Sect. 4 with the proposed model structure given in Sect. 5. Section 6, lists the criteria used in assessing the models. Finally, the results of the developed models are presented in Sect. 7 with concluding comments in Sect. 8.

2 Artificial Neural Network

ANN, a widely parallel distributed processor, has the apt to identify functions, examples or patterns so long as it is trained with prior knowledge. Basically, ANNs have emerged to simulate biological neural systems-particularly the human brain. The fundamental structure of the ANN system consists of a number of interconnected processing elements, referred to as neurons, which are organized in the input, output, and hidden layers. The neurons are joined to each other through a set of synaptic weights. Each neuron receives inputs from a single or multiple neurons, processes it through a specific activation function, and accordingly generates a transformed output to other neurons or external outputs. Although every single neuron performs its task somewhat incompletely or at a slow pace, jointly ANN structure can conduct an immense number of functions efficiently. To create an optimal design for an ANN with good fitting ability, there is a need to create a suitable configuration for the network, mainly in regards to hidden neurons, type of transfer function, number of input variables and the learning algorithm used to adjust the network's weights. Actually, when the number of model parameters increases, this favors the learning over the network and therefore preferable fitting. The learning algorithm may be the most important factor among all in identifying the model, as it is an essential process for updating the network weights and estimating the model parameters that fit with the given data set so that the target function is met. ANN continually updates its weights during the learning process until sufficient knowledge is acquired. When the learning process is completed, it is needful to assess the network's generalization capability using unknown samples of the problem. The importance of learning algorithms has spurred the development of many learning algorithms that are looking for an optimal computational effort that allows for finding optimal quality solutions. Several superb features of ANN have made it a potent computational tool to address a wide range of problems in a variety of areas [6, 7]. ANNs have an ability to learn from unseen examples that have not been formulated, viewed or used. In this context, ANN can be treated as a multivariate nonlinear statistical method and as a universal approximator to approximate nonlinear functions with the desired accuracy.

2.1 Multi-Layer Perceptron

Multi-Layer Perceptron (MLP), as a global approximator of functions, is the most familiar kind of ANN. It became prevailing with the evolution of the Back-Propagation (BP) learning algorithm [8]. The training process of MLP relies on an optimization scheme that looks for the best set of network parameters, or specifically the weights, in relation to input and output patterns to be fit by ANN, referred to as a supervised learning scheme. The MLP network is organized into three layers; the input layer, hidden layer, and output layer. The ANN is expanded when hidden neurons are added to a hidden layer, one by one until the ANN model is capable of achieving its functionality with the lowest possible error. This depends principally on the problem's complexity that is allied to the complexity of the input-output datasets. The BP algorithm consists mainly of two phases. The first layer is ordinarily referred to as the forward pass and the second layer is known as the backward pass. In the forward process: an external input vector $X = (x_1, x_2, \cdots, x_n)$ with dimension n is initially fed to the input neurons of the input layer; then the outputs from the input neurons are fed to the hidden neurons of the hidden layer; finally, the outputs of hidde the n layer are presented to the output neurons of the network, leading to output $y$in the output layer where the network weights are all stationary. In the backward process, the weights are fine-tuned on the basis of the error between the true and the in demand outputs. The prefatory step in training ANNs is to initialize the weight vector $\vec{w}$. During the computation of the forward process, the weight vector is adjusted until it reaches the desired behavior. The output y is assessed to measure the network's performance; if the output is not desirable, the weights have to be iteratively adapted in terms of input patterns. In supervised learning, the goal is to generate an output approximation with coveted patterns of input-output samples as described in Eq. 1.

$$\vec{T}^k = \left\{ \left(x^k \in \mathbb{R}^N, d^k \in \mathbb{R}^M \right) \right\},\ k = 1, 2, \cdots, p \tag{1}$$

where $\vec{T}^k$ are the training samples, x is an input pattern, d is the desired response, N and M are the number of samples in the input and output patterns.

The requirement is to design and analyze the parameters of the network model so that the actual output y^k due to x^k is statistically close to the required degree d^k for all k. The MLP weights can be updated using the BP algorithm. The use of BP algorithm to adjust the ANN's weights may stick in a local minimum, and further, it might not be able to solve non continuous problems. Thence, it may be better to pay attention to other learning algorithms that can address non nonlinear problems, which are critical to achieving high level of performance in solving complex problems. The vector $\vec{w}$ is updated through the learning process of the MLP-type ANN until an error criterion asthe one defined in Eq. 2 is converged to an appropriate value.

$$e = \frac{1}{M \cdot p} \sum_{i=1}^{M} (y_i(x_k, \vec{w}) - d_{ik})^2 \tag{2}$$

Where: y_i is the i^{th} output overall p pattern samples, and d_{ik} is the desired result. The MLP-ANN can be represented mathematically as stated in Eq. 3 [9]:

$$\hat{y}_i(t) = g_i[\varphi, \theta] = F_i \left[\sum_{j=1}^{n_h} W_{i,j} f_j \left(\sum_{l=1}^{n_\varphi} \omega_{j,l} \varphi_l + \omega_{j,0} \right) + W_{i,0} \right] \tag{3}$$

where $\hat{y}_i(t)$ is the output signal at time sample t, g_i is the function recognized by the ANN model and θ identifies the weight parameter vector, which includes all the tunable parameters of the network (weights $\omega_{j,l}$ and biases $W_{i,j}$). Here, MLP is trained using the BP algorithm so that the output $\hat{y}$ corresponds to the input φ when an objective criterion as introduced later is met.

3 Genetic Programming

GP is an evolutionary-based algorithm with a global search potential proposed by Koza in 1992 [10]. GP is among the most well-known algorithms under evolutionary algorithms or nature-inspired algorithms. It is inspired based on the biological evolutionary ideas of natural selection which is capable of finding solutions for a broad variety of real problems through automatically evolving randomly generated models. The relations are arrived at out of an evolutionary search process, with an assortment of potentialities for the real algorithm concerned, for instance, a plan, an expression, a formula, a decision tree, a control strategy or problem-based learning model [11]. The evolutionary process of GP in this context commences by generating a population of individuals at random, each representing a computer program. This is followed by an evaluation of an evaluation metric measure (i.e., the fitness function) of the program with regard to its capacity to reach a solution. The fitness addresses how an individual fits an environment; it is a criterion for selecting individuals who are interested in generating a new population. Programs which are especially fit are chosen for recombination based on the fitness value to create a new population using genetic operators, including selection, crossover besides the mutation operator. The evolutionary process is repeated until a passible solution is reached, or the number of predefined runs is exceeded. The fitness function is recalculated inside each iteration loop until convergence. The programs can be perceived as a syntax tree, where a branch node is a component from a function group, which in turn may accommodate arithmetic functions, trigonometry, and logic operators with at least one argument [10]. The following steps are involved iteratively in the GP evolutionary process until the convergence process is achieved:

1. **Selection step**: some individuals (computer programs) are picked for reproduction using a defined selection procedure. Selection mechanisms may take, for example, one of the following forms:

   - Roulette wheel, where the likelihood that an individual is picked relies on its normalized fitness value [11];
   - Ranking, Which depends on the order of fitness values of individuals [12];
   - Tournament, where individuals are sampled from the population, where the individual with the highest fitness is picked out provided that there is more than one individual [11];
   - Elitism copies the best individuals in the following generation, where it can enhance performance by eschewing the loss of fit individuals [12];

2. **Creation step**: new individuals are produced through the use of reproduction operators, which typically involve the crossover and mutation operators. These operators accomplish random soft changes to create individuals. Crossover is a process that produces two new individuals through a probabilistic swap of genetic information between two randomly selected parents, facilitating a global search for the best individuals in operation. Mutation is an operator that prompts a slight probabilistic change in the genetic structure, resulting in a local search. It selects one individual that commences by choosing a point at random within the tree, and

then it supersedes a function or a terminal set with the same kind of element. Old individuals are replaced by the new individuals created by the reproduction operators in order to build a new generation.

3. **Evaluation step**: This process continues to iterate until an optimal solution based on fitness metric is achieved, or the number of generations is exceeded.

There is an extensive literature on GP for solving a broad range of real complex problems and, more recently, a number of studies have been reported on the success of GP for solving several problems in crucial areas such as software engineering, image processing, manufacturing and statistical modeling [13, 14].

4 Test/Debug Data

To evaluate the accuracy of the developed MLP-ANN and GP models, extensive experiments were conducted on test datasets for two different projects, namely, project A and project B. The collected data consists of 200 modules with each having a one-kilo line of code of FORTRAN for real-time control application [15]. A test/debug data set for project A consists of 111 measurements of test instances (*D*), real detected faults (*F*), number of test workers (*TW*) as given in Table 1.

A test/debug dataset for project B contains 46 measurements as shown in Table 2 [15]. The available measurements are limited in this case. This represents a challenge for traditional modeling methods.

5 Proposed Models for Test Workers Estimation

Two types of models were explored for projects A and B as aforementioned. The first model was developed using MLP-ANN and the second model was formed using GP. Here, a new model structure that can help estimating the number of test workers during the software testing process was proposed. The available data set includes the date of test *d*, the observed number of faults *F* and the number of test workers *y*. The proposed model structure is given in Eq. 4.

$$y = f\left(\frac{\partial F}{\partial t}, d(n-t)\right) \tag{4}$$

Where:$\frac{\partial F}{\partial t}$ is the rate of change of the faults as a function of time t, $t = 1, 2, \cdots, n$ and n is the expected day to finish the software testing. We will rename this attribute as x_1.$d(n-t)$ is the countdown to the end day of testing. We will rename this attribute x_2.

6 Model Evaluation Criteria

In order to verify the performance of the developed MLP-ANN and GP models, we have explored a number of performance assessment functions, including:

- Correlation Coefficient (R^2):

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2} \tag{5}$$

- Mean absolute error (MAE):

$$\mathrm{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y - \hat{y}| \tag{6}$$

Table 1. Test/debug dataset for project A.

D	*F*	*TW*	*D*	*F*	*TW*	*D*	*F*	*TW*	*D*	*F*	*TW*
1	5	4	29	2	6	57	2	4	85	0	2
2	5	4	30	5	6	58	3	4	86	0	2
3	5	4	31	4	6	59	2	4	87	2	2
4	5	4	32	1	6	60	7	4	88	0	2
5	6	4	33	4	6	61	3	4	89	0	2
6	8	5	34	3	6	62	0	4	90	0	2
7	2	5	35	6	6	63	1	4	91	0	2
8	7	5	36	13	6	64	0	4	92	0	2
9	4	5	37	19	8	65	1	4	93	0	2
10	2	5	38	15	8	66	0	3	94	0	2
11	31	5	39	7	8	67	0	3	95	0	2
12	4	5	40	15	8	68	1	3	96	1	2
13	24	5	41	21	8	69	1	3	97	0	2
14	49	5	42	8	8	70	0	3	98	0	2
15	14	5	43	6	8	71	0	3	99	0	2
16	12	5	44	20	8	72	1	3	100	1	2
17	8	5	45	10	8	73	1	4	101	0	1
18	9	5	46	3	8	74	0	4	102	0	1
19	4	5	47	3	8	75	0	4	103	1	1
20	7	5	48	8	4	76	0	4	104	0	1
21	6	5	49	5	4	77	1	4	105	0	1
22	9	5	50	1	4	78	2	2	106	1	1
23	4	5	51	2	4	79	0	2	107	0	1
24	4	5	52	2	4	80	1	2	108	0	1
25	2	5	53	2	4	81	0	2	109	1	1
26	4	5	54	7	4	82	0	2	110	0	1
27	3	5	55	2	4	83	0	2	111	1	1
28	9	6	56	0	4	84	0	2			

Table 2. Test/debug dataset for project B.

D	*F*	*TW*	*D*	*F*	*TW*	*D*	*F*	*TW*	*D*	*F*	*TW*
1	2	75	13	3	78	25	1	15	37	3	5
2	0	31	14	4	48	26	7	31	38	0	27
3	30	63	15	4	75	27	0	1	39	0	6
4	13	128	16	0	14	28	22	57	40	0	6
5	13	122	17	0	4	29	2	27	41	0	4
6	3	27	18	0	14	30	5	35	42	5	1
7	17	136	19	0	22	31	12	26	43	2	6
8	2	49	20	0	5	32	14	36	44	3	5
9	2	26	21	0	9	33	5	28	45	0	8
10	20	102	22	30	33	34	2	22	46	0	2
11	13	53	23	15	18	35	0	4			
12	3	26	24	2	8	36	7	8			

- Root mean square error (RMSE):

$$\mathrm{RMSE} = \sqrt{\frac{1}{n}\sum\nolimits_{i=1}^{n}(y-\hat{y})^2} \tag{7}$$

- Relative absolute error (RAE):

$$\mathrm{RAE} = \frac{\sum_{i=1}^{n}|y-\hat{y}|}{\sum_{i=1}^{n}|y-\bar{y}|} \tag{8}$$

- Root relative squared error (RRSE):

$$\mathrm{RRSE} = \sqrt{\frac{\sum_{i=1}^{n}(y-\hat{y})^2}{\sum_{i=1}^{n}(y-\bar{y})^2}} \tag{9}$$

where y is the actual number of test workers, $\hat{y}$ is the estimated value and $\bar{y}$ is the mean of the signal y using n measurements.

7 Experiments and Results

We generated the required attributes $x_1$and$x_2$from the available data sets of Projects A and B to train an ANN and develop a GP mathematical model for test worker estimation. For MLP-ANN, one hidden layer was used while the rest of the parameters are tuned as shown in Table 3. The tuning parameters of GP are set as shown in Table 4.

Table 3. MLP Parameters.

Parameter	Value
Parameter	Value
Architecture	MLP
Training method	Back-propagation
Hidden Layers	1
hidden neurons	30
Learning rate	0.9
Maximum epochs	300

Table 4. GP regression parameters.

Parameter	Value
Population size	500
Maximum tree depth	4
Selection mechanism	Tournament
Tournament size	10
Mutation probability	0.14
Crossover probability	0.84
Function operators	[+, −, ×]
Cross-validation	10-fold

The observed and estimated number of workers calculated for the test instances and the real detected faults for Project A based on ANN is shown in Fig. 3 and its convergence is shown in Fig. 4. For Project B, the observed and estimated number of workers and the convergence of ANN are shown in Figs. 5 and 6, respectively.

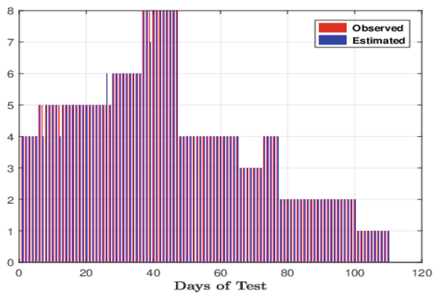

Fig. 3. Project A: observed and estimated number of test workers using MLP.

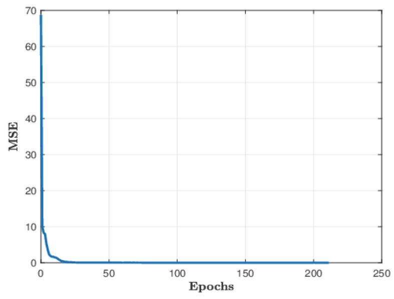

Fig. 4. Project A: convergence of the MLP model

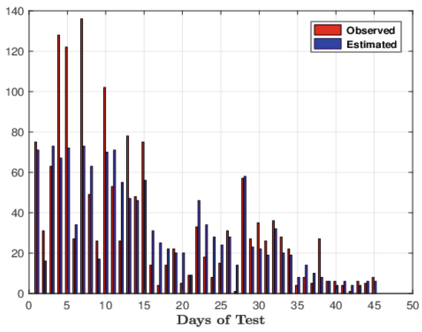

Fig. 5. Project B: observed and estimated number of test workers using MLP.

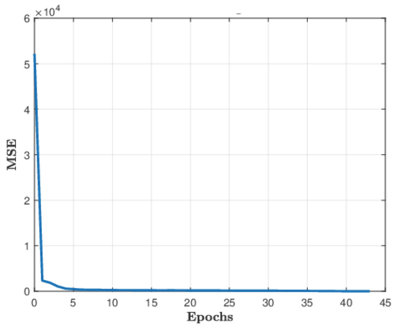

Fig. 6. Project B: convergence of the MLP model.

The GP models for projects A and B are shown respectively in Eqs. 10 and 11.

$$y_A = 0.00528x_1(x_1 + x_2)^2 - 0.0895x_1x_2 - 3.56 \times 10^{-4}x_1^2x_2 - 0.0886x_1^2 + 8.95 \times 10^{-4}x_2^3 - 5.03 \times 10^{-6}x_1^2x_2(x_1 + x_2)^2 + 9.33 \quad (10)$$

$$y_B = 0.0614x_1^2 - 6.72 \times 10^{-4}x_1^2x_2 + 0.00202x_2^2 - 1.63 \times 10^{-5}x_2^3 - 6.38 \times 10^{-7}x_1x_2(x_1 + 5.0)(x_1 - x_2) + 1.03 \quad (11)$$

The observed and predicted numbers of test workers for projects A and B based GP are shown in Figs. 7 and 8, respectively. The evaluation results of MLP and GP models are shown in Table 5. The results of these models are very competitive with a slight superiority for MLP in one case and GP in the other case. GP still has merit of creating efficient mathematical models which are easier to value than MLP models.

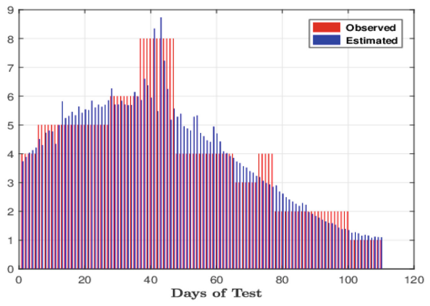

Fig. 7. Project A: observed and estimated number of test workers using GP.

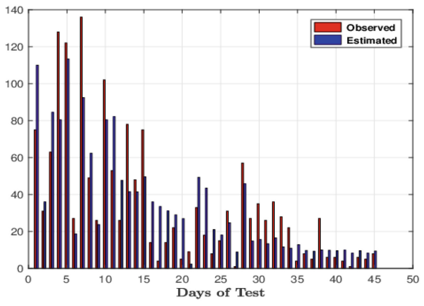

Fig. 8. Project B: observed and estimated number of test workers using GP.

Table 5. Evaluation results of MLP-ANN and GP models.

Criterion	Project A		Project B	
	MLP	GP	MLP	GP
Correlation coefficient	0.99531	0.90161	0.84726	0.84228
Mean absolute error	0.036364	0.036364	13.044	14.778
Root mean squared error	0.19069	0.8528	19.589	18.598
Relative absolute error	2.3918%	34.681%	50.355%	57.046%
Root relative squared error	9.7244%	43.489%	56.775%	53.905%
Total Number of Instances	111	111	46	46

8 Conclusions

The proposed work has demonstrated the use of two computational techniques, namely, Multilayer Perceptron Artificial Neural Network and Genetic Programming methods, to model the relationship between the number of test workers and the measured faults in software to build two prediction models. In this context, the developed models utilized the expected day to finish testing and the rate of change of faults as inputs to the models. Two case studies were presented, and several evaluation criteria were conducted to validate the performance of the proposed models. All evaluation measures have reported a high level of performance, on the basis of the satisfactory predication estimates obtained, suggesting that the presented MLP and GP models are highly accurate, learned the dynamic relationships between the inputs and output successfully. The use of GP and ANN to estimate the number of test workers using software faults is an exciting direction for future research. Further work is needed to assess the suitability of the proposed models to other test instances.

References

1. Yoon, I.C., Sussman, A., Memon, A., Porter, A.: Effective and scalable software compatibility testing. In: The 2008 ISSTA, pp. 63–74. ACM (2008)
2. Sheta, A.F.: Estimation of the COCOMO model parameters using genetic algorithms for NASA software projects. J. Comput. Sci. **2**(2), 118–123 (2006)
3. Sheta, A.F., Ayesh, A., Rine, D.: Evaluating software cost estimation models using particle swarm optimization and fuzzy logic for NASA projects: a comparative study. IJBIC **2**(6), 365–373 (2010)
4. Sheta, A.F., Kassaymeh, S., Rine, D.: Estimating the number of test workers necessary for a software testing process using artificial neural networks. IJACSA **5**(7), 186–192 (2014)
5. Sheta, A.F., Abdel-Raouf, A.: Estimating the parameters of software reliability growth models using the Grey Wolf optimization algorithm. Int. J. Adv. Comput. Sci. Appl. **1**(7), 499–505 (2016)
6. Braik, M., Sheta, A., Arieqat, A.: A comparison between GAs and PSO in training ANN to model the TE chemical process reactor. In: The AISB 2008 Symposium on Swarm Intelligence Algorithms and Applications, vol. 11, pp. 24–30 (2008)
7. Sheta, A.F., Braik, M., Al-Hiary, H.: Identification and model predictive controller design of the tennessee eastman chemical process using ANN. In: The International Conference on Artificial Intelligence (ICAI 2009), vol. 1, pp. 25–31 (2009)
8. Rumelhart, D.E., Hinton, G.E., Williams, R.J.: Neurocomputing: foundations of research. In: Anderson, J.A., Rosenfeld, E. (eds.) Learning Representations by Back-propagating Errors, pp. 696–699. MIT Press, Cambridge (1988)
9. Al-Hiary, H., Braik, M., Sheta, A., Ayesh, A.: Identification of a chemical process reactor using soft computing techniques. In: The 2008 International Conference on Fuzzy Systems within the 2008 IEEE World Congress on Computational Intelligence (WCCI 2008), pp. 845–853 (2008)
10. Koza, J.R.: Genetic Programming II, Automatic Discovery of Reusable Subprograms. MIT Press, Cambridge (1992)
11. Dimitriu, R.C., Bhadeshia, H., Fillon, C., Poloni, C.: Strength of ferritic steels: Neural networks and genetic programming. Mater. Manuf. Process. **24**(1), 10–15 (2008)
12. Naudts, B., Kallel, L.: A comparison of predictive measures of problem difficulty in evolutionary algorithms. IEEE Trans. Evol. Comput. **4**(1), 1–15 (2000)
13. Faris, H., Sheta, A.: Identification of the Tennessee Eastman chemical process reactor using genetic programming. Int. J. Adv. Sci. Tech. **50**, 121–140 (2013)

14. Sheta, A.F., Faris, H., Öznergiz, E.: Improving production quality of a hot-rolling industrial process via genetic programming model. Int. J. Comput. Appl. Technol. **49**(3/4), 239–250 (2014)
15. Tohma, Y., Tokunaga, K., Nagase, S., Murata, Y.: Structural approach to the estimation of the number of residual software faults based on the hyper-geometric distribution. IEEE TSE. **15**(3), 345–355 (1989)