IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received December 10, 2019, accepted December 30, 2019, date of publication January 6, 2020, date of current version January 15, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.2964321

Enhanced Binary Moth Flame Optimization as a Feature Selection Algorithm to Predict Software Fault Prediction

IYAD TUMAR[1], (Member, IEEE), YOUSEF HASSOUNEH[2], HAMZA TURABIEH[3], AND THAER THAHER[4]

[1]Electrical and Computer Engineering Department, Birzeit University, Ramallah 00970, Palestine
[2]Department of Computer Science, Birzeit University, Ramallah 00970, Palestine
[3]Department of Information Technology, Taif University, Taif 21431, Saudi Arabia
[4]Department of Computer Science, Al-Quds University, Jerusalem 00970, Palestine

Corresponding author: Hamza Turabieh (h.turabieh@tu.edu.sa)

ABSTRACT Software fault prediction (SFP) is a complex problem that meets developers in the software development life cycle. Collecting data from real software projects, either while the development life cycle or after lunch the product, is not a simple task, and the collected data may suffer from imbalance data distribution problem. In this research, we proposed an Enhanced Binary Moth Flame Optimization (EBMFO) with Adaptive synthetic sampling (ADASYN) to predict software faults. BMFO is employed as a wrapper feature selection, while ADASYN enhances the input dataset and address the imbalanced dataset. Converting MFO algorithm from a continues version to the binary version using transfer functions (TFs) from two different groups (S-shape and V-shape) is investigated in this work and proposed an EBFMFO version. Fifteen real projects data obtained from PROMISE repository are employed in this work. Three different classifiers are used: the k-nearest neighbors (k-NN), Decision Trees (DT), and Linear discriminant analysis (LDA). The reported results demonstrate that the proposed EBMFO enhances the overall performance of classifiers and outperforms the results in the literature and show the importance of TF for feature selection algorithms.

INDEX TERMS Software fault prediction, feature selection, binary moth flame optimization, adaptive synthetic sampling, classification.

I. INTRODUCTION

The process of developing good software consists of several stages, such as software requirements, analysis, design, implementation, testing, and documentation. The test phase is an important stage that enhances the quality of the software and reduces the total cost. In practice, testing is performed either as a linear approach (i.e., waterfall) or cyclical (i.e., incremental, iterative, agile) models. Finding or predicting faults called Software fault prediction (SFP). SFP detects either clear or hidden fault-prone modules in advance before new software versions being developed. SFP process determines the efficiency of the new software based on several factors, such as historical fault datasets, user comments and predefined software metrics [1], [2]. Developing software based on incremental delivery (known as Agile Software Development (ASD) methodology) will minimize the development time, and thus delivering the new software before the deadline approaches in addition to reducing the gap between developers and business owners [3]. However, repaid software development leads to faults. As a result, SFP becomes a mandatory step in order to predict faults and satisfy end users. This process helps in reduces the costs needed to finish a project, and thus in improving the subsequent versions.

Software quality assurance (SQA) aims to control the software development lifecycle (SDLC) to ensure that the current system meets the expectations. SQA consists of several applications such as code walkthroughs, software testing, and SFP [4], [5]. SFP models predict the expected faults during development stage, which enhances the overall software quality. These models are developed based on either software metrics (i.e. change or file status metrics) or fault datasets (aggregated from previous versions of similar projects). Such models are helpful when resources of the project are not adequate, or the

The associate editor coordinating the review of this manuscript and approving it for publication was Baoping Cai.

system is quite large and difficult to test. In general, building SFP models depends on three factors: software metrics, soft computing (SC), and machine learning (ML) algorithm, and techniques [1]. The process of developing the software metrics model is related to collecting metrics data to predict the faults [6]. This approach does not work smoothly with different projects or different versions [7]. So, researchers use software change metrics (i.e. historical changes) to overcome this drawback and build an accurate SFP model. However, this approach is considered time-consuming and impractical with complex systems.

ML algorithms are the heart of data science that used successfully to solve complex problems either in the industrial or research world. The performance of ML algorithms depends on several factors such as data dimensionality, data representation, and ML algorithms. High data dimensionality that has irrelevant, noisy, and redundant data will reduce the overall performance of ML classifiers. Extracting these features will reduce the dimensionality and enhance the performance of ML algorithm [8]. Generally, reducing the data dimensionality is performed using feature selection (FS) algorithms [9]. FS algorithms remove noisy, irrelevant, and redundant data without reducing the performance of ML algorithms. Moreover, FS enables developers and researchers to understand the data itself and focus on the most valuable features [10]. Several research papers in the literature reported that FS algorithms are able to enhance the performance of SFP systems [11].

Selecting the best features for SFP problem is challenging process since projects have different requirements and different development procedures. The high dimensionality of data with redundant and noisy data will increase the learning time for ML classifiers and will not guarantee to achieve a high-quality model. So, the motivation of this work is to propose an intelligent FS algorithm based on MFO that is able to address all issues related to high dimensionality data that will enhance the overall performance of SFP model.

The rest of this paper is organized as follows: a literature review of related works for SFP methods and FS algorithms is presented in Section II. In Section III, the proposed intelligent approach is discussed deeply. In Section IV, set of TFs with their mathematical models are presented. The imbalanced data problem is addressed in Section V. Sections VI and VII discussed the classification methods and evaluation criteria, respectively. Section VIII presents the obtained results and their analysis. Finally, Section IX concludes the obtained results and future works for this research paper.

II. REVIEW OF RELATED WORKS

SFP problem has been tackled using several algorithms. These algorithms mainly focus on machine learning algorithms. Researchers applied feature selection algorithms incorporating a selected classifier(s) to enhance the overall performance. In the next subsections, we will explore the machine learning and feature selection algorithms that solved SFP problem.

A. SOFTWARE FAULT PREDICTION

ML algorithms show promising performance in solving SFP problem. Several algorithms are used such as Naive Bayes (NB) [12], Multilayer Perceptron (MLP) [13], Case-based Reasoning (CR) [14], Artificial Neural Networks [15], Deep learning methods [1], Support Vector Machine (SVM) [16], Bayesian Networks (BN) [17], Decision Trees (DT) [18], Multinomial Logistic Regression (MLR) [13] and Logistic Regression (LR) [17]. Several public bechmarck datasets are available online, and researchers employed their proposed algorithms and compare the obtained results with the literature.

One of the most well-known public datasets is NASA repository, that has been used by researchers in this field. For example, Cahill *et al.* [19] introduced a ranking approach called Rank Sum to examine the achieved results for SFP problem. The authors applied SVM and NB machine learning algorithms, and NB results outperform SVM. Carrozza *et al.* [13] examine five ML algorithms (i.e., MLR, BN, NB, SVM, abd DT) over several datasets from NASA repository. The performance of MLP and SVM outperform other algorithms. Moreover, the authors proposed new metric criteria for complex systems that have Mandelbugs. Malhotra [20] investigates the performance of multi ML and LR algorithms on another public dataset obtained the PROMISE repository. Experimental results show that DT algorithm outperform all examine algorithms. Khoshgoftaar *et al.* [21] proposed a hybrid approach called multi-strategy classifier (RB2CBL), which hybridized with Rule-based (RB) model with two different case-based learning (CBL). The authors employed genetic algorithms to optimize CBL parameters. The obtained results show that RB2CBL outperform standard BR model. Rathore and Kumar [22] tackle two different datasets from PROMISE repository and Eclipse bug data repository using two ensemble learners methods, linear regression based combination rule (LRCR) and Gradient boosting regression based combination rule (GRCR). The authors employed two evaluation criteria evaluate the obtained results, Average Absolute Error (AAE) and Average Relative Error (ARE), and GRCR model outperforms LRCR one.

Erturk and Ebru [23] solve SFP problem based on an iterative hybrid approach between Fuzzy Inference System (FIS) and Artificial Neural Network (ANN). The FIS is employed at the beginning of the project where no historical data available, while ANN is employed once some historical data available. Shatnawi [24] investigate the performance of different set of ML algorithms based on receiver operating characteristic (ROC). The author proposed a threshold value to determine that project has a fault or not. Choudhary *et al.* [25] study the importance of change metrics with code metrics to improve the overall performance of SFP models. The authors examine their proposed approach on a set of Eclipse projects for code metrics and change metrics are extracted from the GIT repositories, the obtained results improve the overall performance of SFP models.

Sharma and Chandra [26] review several machine learning and conventional algorithms to solve SFP problem and highlight the importance of SFP in reality.

B. FEATURE SELECTION

FS is NP-hard search problem [27], [28] that tries to determine the optimal number of features or attributes from the original dataset without losing the main functionality of the original dataset. The problem complexity increased exponentially based on the number of features. Therefore, researchers employed heuristic search to enhance to obtained results and the computational time for large problems. In general, FS algorithms perform two tasks: (i) Find the minimal number of features that represent the original dataset, and (ii) evaluate the selected features based on a predetermined fitness function. FS algorithms evaluate selected features based on two approaches: *filter* and *wrapper*. Filter approach evaluates the selected features based on the relation between features, while the wrapper approach uses a learning algorithm to evaluate the selected features. Based on execution time, filter approach is faster than wrapper one. However, the wrapper approach is more accurate than filter approach [29].

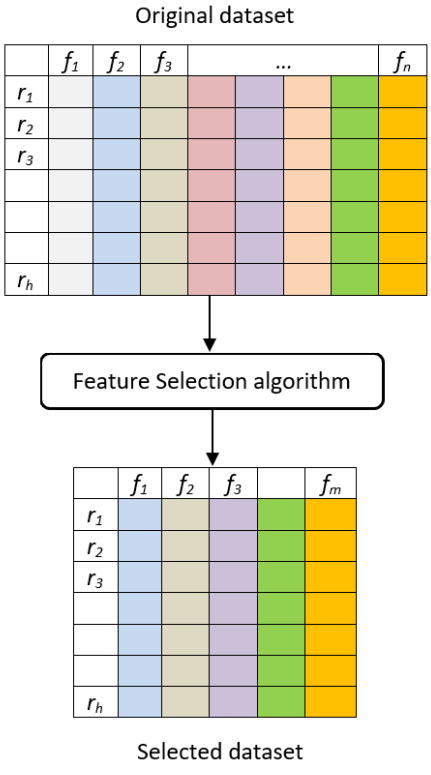

FIGURE 1. A pictorial diagram of the feature selection.

A binary representation is the simplest approach to tackle FS problems, where 0 means the feature is not selected, while 1 means that the feature is selected. Figure 1 shows a pictorial diagram for a dataset of n features. Applying FS algorithm will select m features from the original n features, where $m \leq n$.

Several algorithms have been proposed as features selection algorithms. These algorithms are classified into two groups such as exact algorithms and meta-heuristic search algorithms [30]. Meta-heuristic algorithms show a higher performance compared to the exact method for complex problems [1]. In general, meta-heuristic algorithms are classified into two groups: (i) single-solution algorithms (S-based), (e.g. Tabu Search (TS) [31], Great Deluge (GD) [32], and Simulated Annealing (SA) [33]), and (ii) population-based algorithms (P-based), (e.g. Artificial Bee Colony algorithm (ABC) [30], Genetic Algorithms (GA) [34], Harris hawks optimization [35], Moth Flame Optimization (MFO) [36], Particle Swarm Optimization (PSO) [37], and Whale Optimization Algorithm [38]). The S-based algorithms focus on exploitation process, while P-based algorithms focus on exploration process. In general, P-based algorithms are able to research more areas in the search space and achieve more accurate results compared to S-based method. However, S-based method execution time is less than P-based one.

III. PROPOSED APPROACH

In this work, we employed a wrapper approach by adopting the binary version of MFO algorithm to tackle SFP problem. Figure 2 explores the methodology of the proposed algorithm. At each iteration, a cross-validation process with k = 10 is evaluated. The algorithm begins by employing MFO on the SFP dataset. At each iteration, the selected dataset is divided using cross-validation approach. An adaptive synthetic sampling approach is used to address the imbalanced dataset for the training dataset. This process will create oversampling training dataset. Three different classifiers (i.e. kNN, DT, and LDA) are used to build an SFP model and evaluated based on Kfold = 10. The proposed approach stops once achieve the optimal solution based on the fitness function or reach maximum number of iterations. The following subsections demonstrate the proposed model.

A. MOTH FLAME OPTIMIZATION

MFO is a swarm optimization algorithm that was firstly introduced by Mirjalili in 2015 [39]. Moth is an insect that is related to the butterflies family; where its main activities start at night. The basic idea for MFO comes from the exploration process of moths while searching for light in nature, which is called transverse orientation while traveling toward light at night, depending on the light that comes either from the moon or man-made light. The position of moths is controlled based on a fixed movement angle concerning the incomming light. The moths move in a spiral shape and try to keep a similar angel for the man-made light. This flying approach will create a deadly spiral fly path for moths [39]. Figure 3 demonstrates the moths spiral flying path toward a man-made light.

B. MFO MATHEMATICAL FORMULATION

MFO is a population-based algorithm, where each moth presents a candidate solution in the search space for

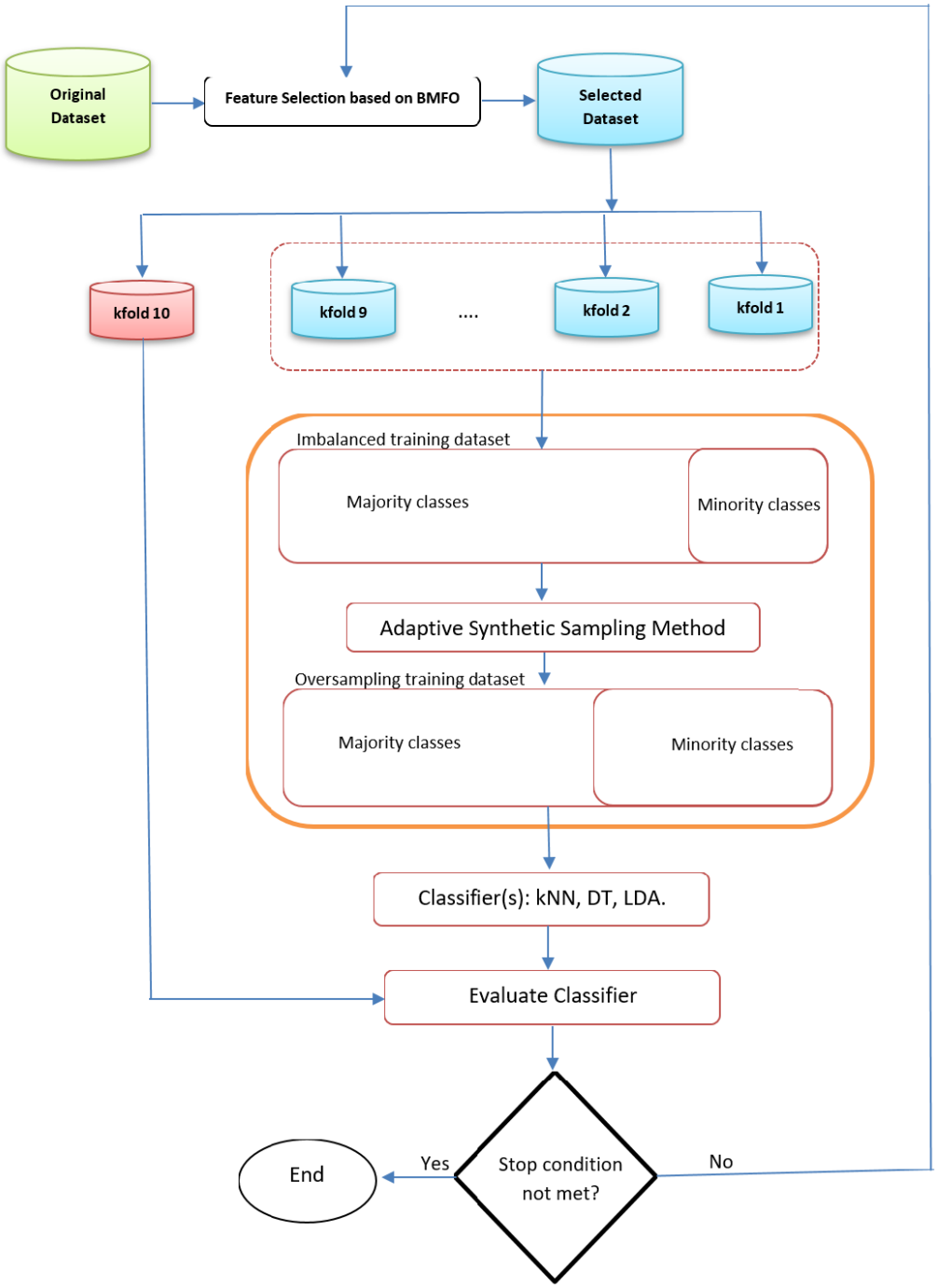

FIGURE 2. Proposed methodology.

the problem. The following matrix demonstrates the moths solutions.

$$M = \begin{bmatrix} m_{1,1} & m_{1,2} & . & . & m_{1,d} \\ . & . & . & . & . \\ . & . & . & . & . \\ . & . & . & . & . \\ m_{n,1} & m_{n,2} & . & . & m_{n,d} \end{bmatrix}$$

where d indicates the problem dimension (i.e. number of variables), and n refers to the number of moths (Solutions).

The best positions in the search space presented as a set of flames. The following matrix demonstrates the set of flames, which is similar to matrix of moth.

$$F = \begin{bmatrix} F_{1,1} & F_{1,2} & . & . & F_{1,d} \\ . & . & . & . & . \\ . & . & . & . & . \\ . & . & . & . & . \\ F_{n,1} & F_{n,2} & . & . & F_{n,d} \end{bmatrix}$$

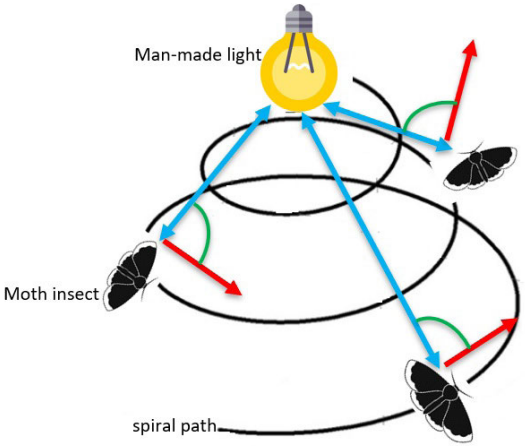

FIGURE 3. Spiral flying path for moths around close light source.

where both d and n presents the dimension of the search space and moths, respectively. Moths and flames are solutions in the search space. However, the main difference between them is the updating process. Moths resent the actual search agents that explore the search space. Each moths exploit around a flame and update its position. This approach enables moths to balance between exploration and exploitation process while search process [39]. Equation 1 presents the updating process based on the flames, where M_i represents the i^{th} moth and F_j is the j^{th} flame. Equation 2 presents the logarithmic spiral function that is used to update moth in the search space, where b is a consist value for control the shape of the logarithmic spiral function, t is a random value between -1 and 1, and D_i represents distance between M_i moths and F_j flame. Equation 3 presents the D_i calculation process. All flames are sorted ascending based on the fitness values in each generation. So, the moths update mechanism for its positions based on the closest best flames (best solutions).

$$M_i = P(M_i, F_j) \tag{1}$$

$$S(M_i, F_j) = D_i \cdot e^{bt} \cdot cos(2\pi t) + F_j \tag{2}$$

$$D_i = |F_j - M_i| \tag{3}$$

In the beginning of MFO algorithm, a predefined number of flames N will be determined. This number of flames will be gradually decreased with more iterations due to the updating mechanism for moths. This process of decrementation will keep a good ratio between the exploitation and exploration [39]. Equation 4 presents the number of flames inside MFO algorithm, where N is a predefined number represents the initial number of flames at first iteration, l represents the actual number of iteration, and T represents the maximum number of iterations. The pseudo-code of MFO is shown in Algorithm 1.

$$FlameNumber = round(N - l \times \frac{N - l}{T}) \tag{4}$$

In binary moths flame optimization (BMFO), each moth (solution) is represented by a binary vector $x = (x_1, x_2, \ldots, x_n), x_i \in \{0, 1\}$. Figure 4 demonstrates the binary representation of moth solution.

Algorithm 1 The Original Pseudo-Code of MFO Algorithm

Input: Total number of moths and iterations (T_{max}).
Output: The best solution and the its fitness value.
Initialize positions of moths $x_i(i = 1, 2, \ldots, n)$
Obtain the fitness of moths.
while (Looping condition is not met) **do**
  Update flame no. based on Eq. (4)
  Define: OM = Fitness Function(M)
  if $(i == 1)$ **then**
    F = sort(M), OF = sort(OM)
  else
    F = sort(M_{t-1}, M_t), OF = sort(M_{t-1}, M_t);
  end if
  for $i = 1 : n$ **do**
    for $j = 1 : d$ **do**
      Update r and t
      Obtain D by Eq. (3) with regard to the related moth
      Update $M(i, j)$ via Eqs. (1) and (2) with regard to the related moth
    end for
  end for
end while
Return the best solution

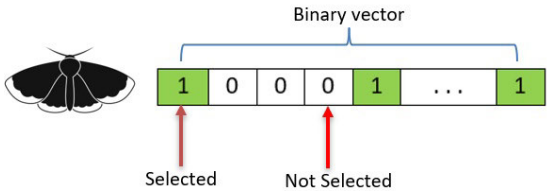

FIGURE 4. Binary presentation for moth solution.

C. ENHANCED BMFO (EBMFO)

To enhance the performance of the exploration and exploitation of MFO algorithm, we modified the moth-flame following strategy and population diversity of MFO using evolutionary population dynamics (EPD). EPD was applied successfully by several researchers to control the population diversity of several meta-heuristic algorithms [40], [41]. The following subsections present both enhancement components.

D. MOTH-FLAME FOLLOWING STRATEGIES

MFO sorted flames in ascending order based on fitness function at the beginning of MFO algorithm, while moths are not sorted. In the original MFO, the first moth follows the fittest best flame (index = 1), while the last moth follows the worst flame that have the similar index as shown in Figure 5. Since there are a large number of flames exists at the beginning of algorithm execution, the probability of moth to follow a flame with worst fitness value is high. This will force moth to move in a direction far away to the source light (best solution) if the last moth fitness value is better than the last flame

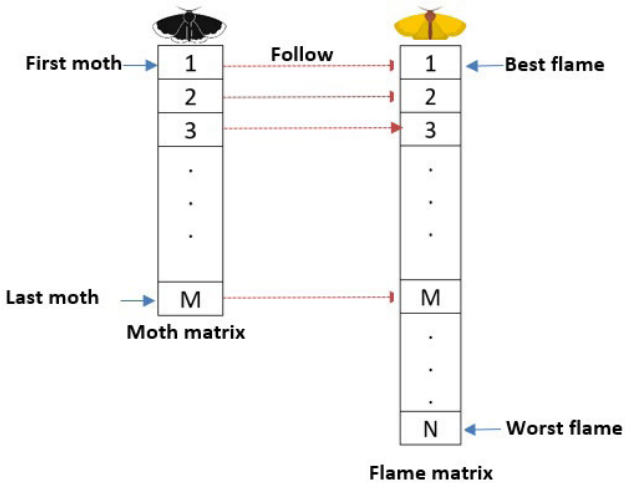

FIGURE 5. Original moth-flame following strategy (ascending order).

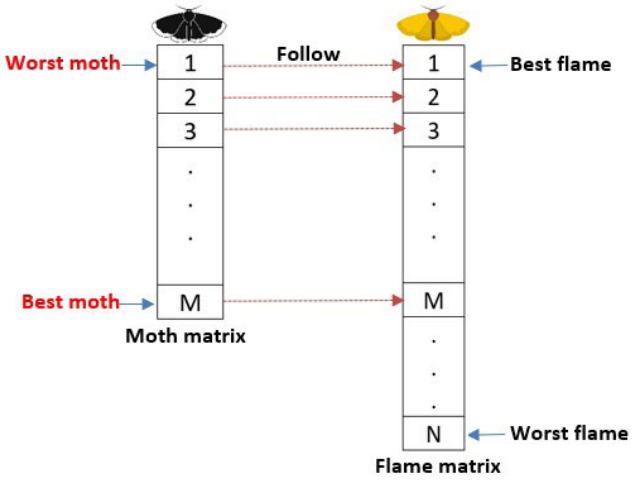

FIGURE 6. Enhanced moth-flame following strategy (descending order).

fitness value. As a result, a set of iterations will not improve the moths fitness values, which increase the computational time. To address this problem, the moths are sorted in descending order, to force the worst moth to follow the best flame as shown in Figure 6. In this scenario, we make sure that from the first iteration all worst moth moves toward best flame (source light).

After a set of iterations, the number of flames will decrease gradually, and number of moths becomes greater than number of flames. In this case, the original MFO algorithm forces all moths that have index greater than last flame index to follow the last flame as shown in Figure 7. This original scenario will force a large number of moths to follow the worst flame. To address this issue, we proposed that all moths that have index greater than last flame index will select a flame randomly from flame matrix. This process will enhance the exploitation process inside MFO algorithm.

E. POPULATION DIVERSITY BASED ON EPD

To enhance the population diversity and to avoid premature convergence for MFO, an EPD method is employed.

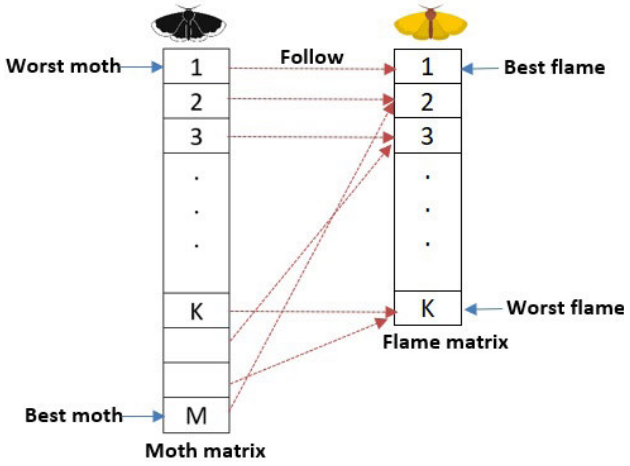

FIGURE 7. Random selection of flame to follow strategy.

All moths are sorted in ascending order. EPD will divide the moths matrix into two halves, Best-half and Worst-half. To keep the population diversity high, the worst-half is replaced by applying a uniform crossover operator between each solution in the worst-half with a randomly selected solution from the best-half. This process will enhance the exploration ratio for MFO algorithm.

IV. TRANSFER FUNCTIONS

MFO is a continuous algorithm in nature, converting MFO to a binary version should utilize the transfer function (TF). To achieve this, we adopted eight different TFs from two groups, S-shaped and V-shaped. Equation 5 explores the probability of updating the process of selecting features from a binary vector, 1 means selected features, while 0 means not selected. Several researchers employed this mechanism to convert continuous algorithms to a binary version [42], [43].

$$T(x_j^i(t)) = \frac{1}{1 + \exp^{-x_j^i(t)}} \tag{5}$$

where the variable x_j^i represents the j^{th} element in x solution in the j^{th} dimension, and t is the current iteration.

Equation 6 presents the updating process for S-shape group for the next iteration.

$$x_i^k(t+1) = \begin{cases} 0 & \text{If } rand < T(v_i^k(t+1)) \\ 1 & \text{If } rand \geq T(v_i^k(t+1)) \end{cases} \tag{6}$$

where $x_i^d(t+1)$ is the i^{th} element at d^{th} dimension in x solution, $T(x_j^i(t))$ represents the probability value that can can be calculated from Equation 5.

Equation 8 explores the updating process for V-shape group for the next iteration, based on the probability values that can be calculated from Equation 7 [44]. The mathematical models for both groups (S-shape and V-shape) are presented in Table 1.

$$T(x_j^i(t)) = |\tanh(x_j^i(t))| \tag{7}$$

$$X_{t+1} = \begin{cases} \neg X_t & r < T(\Delta x_{t+1}) \\ X_t & r \geq T(\Delta x_{t+1}) \end{cases} \tag{8}$$

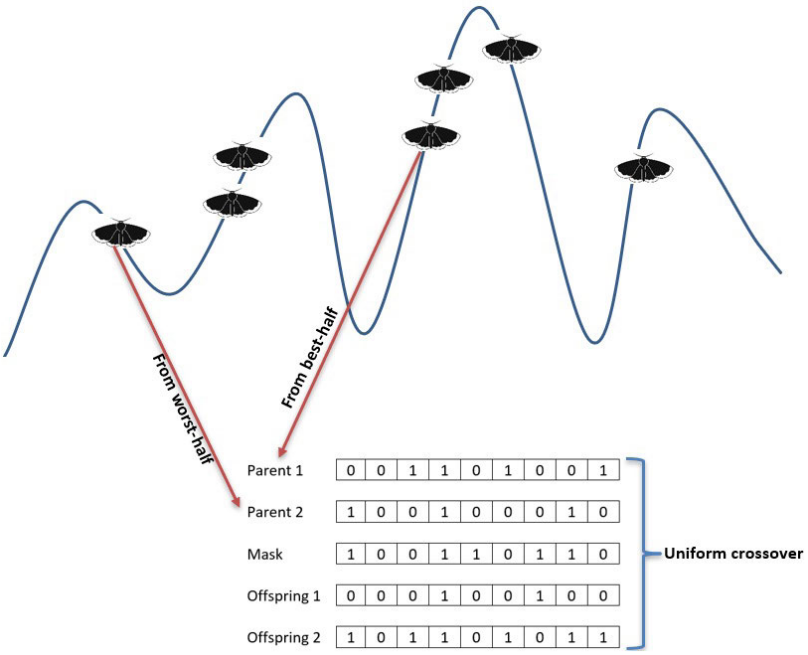

FIGURE 8. EPD process for BMFO algorithm.

TABLE 1. S-shaped and V-shaped transfer functions.

S-shaped family		V-shaped family	
Name	Transfer function	Name	Transfer function
S1	$T(x)=\frac{1}{1+e^{-2x}}$	V1	$T(x)=\lvert\mathrm{erf}(\frac{\sqrt{\Pi}}{2}x)\rvert=\lvert\frac{\sqrt{2}}{\Pi}\int_0^{(\sqrt{\Pi}/2)x}e^{-t^2}dt\rvert$
S2	$T(x)=\frac{1}{1+e^{-x}}$	V2	$T(x)=\lvert\tanh(x)\rvert$
S3	$T(x)=\frac{1}{1+e^{(-x/2)}}$	V3	$T(x)=\lvert(x)/\sqrt{1+x^2}\rvert$
S4	$T(x)=\frac{1}{1+e^{(-x/3)}}$	V4	$T(x)=\lvert\frac{2}{\Pi}\mathrm{arc}\tan(\frac{\Pi}{2}x)\rvert$

V. IMBALANCED DATA LEARNING

The performance of classifier affected by several factors such as number of samples and number of class type. The imbalance problem for collected data occurs when the class of interest (minority class) is very small compared to the normal class (majority class). Intelligent machine learning classifiers usually suffers when the input dataset is skewed toward one class. In reality, most of the collected data suffer from imbalanced data, which reduces the overall performance of classifiers [45].

Several research papers highlighted the imbalanced dataset problem and proposed several methods to address it. In general, there are two main methods to handle imbalanced data: the data perspective and the algorithm perspective [46]. The data perspective rebalances the class distribution based on re-sampling the data space, using either over-sampling or under-sampling instances for minority class or majority class, respectively. The re-sampling methods try to overcome imbalance dataset problem either randomly or deterministically.

One of the most recommended approaches to address imbalanced data are called SMOTE (Synthetic Minority Over-sampling TEchnique), that generates synthetic samples between every positive sample and one of its close sample [46], and Adaptive synthetic sampling (ADASYN), that create a weighted distribution for several minority class based on their difficulty while learning process, several synthetic data is created for minority class which make learning process more easier [47]. The main advantages of ADASYN are reducing the bias toward the minority class and adaptively learning.

VI. CLASSIFIERS METHODS

There are several classifiers learning classifiers in the literature. So, we limit our work to employ only three different classifiers: nearest neighbors (kNN), Linear discriminant analysis (LDA) and decision trees (DT). These classifiers are applied in different domains successfully. The kNN classifiers work based upon the similarity threshold value to classify the dominant class to the nearest group [48]. In this work, we select $k = 5$. LDA is a statistical machine learning approach, which finds an optimal projection by mean of Fisher criterion optimization. LDA employs a scatter approach in each class concerning the overall data average [49]. Decision tree (DT) classifier that works based upon the information-based criteria to set up decision trees [50]. The tree is extended once new valuable information obtained.

VII. EVALUATION METHOD

A. RECEIVER OPERATING CHARACTERISTIC CURVE-AREA UNDER CURVE (AUC)

Classification and prediction problems have several evaluation criteria such as accuracy, precision, and Receiver

TABLE 2. A matrix of the confusion matrix.

		Predicted Class	
		Class = Yes	Class = No
Actual Class	Class = Yes	True Positive (TP)	False Negative (FN)
	Class = No	False Positive (FP)	True Negative (TN)

TABLE 3. AUC values description [51].

AUC Range	Description
$0.00 \leq \text{AUC} < 0.50$	bad classification.
$0.50 \leq \text{AUC} < 0.60$	poor classification.
$0.60 \leq \text{AUC} < 0.70$	fair classification.
$0.70 \leq \text{AUC} < 0.80$	acceptable classification.
$0.80 \leq \text{AUC} < 0.90$	excellent classification.
$0.90 \leq \text{AUC} \leq 1.00$	outstanding classification.

TABLE 4. The selected imbalanced SFP datasets.

Dataset	version	#features #features	#instances #instances	#defective instances	%defective instances
ant	1.7	20	745	166	0.223
camel	1.0	20	339	13	0.038
camel	1.2	20	608	216	0.355
camel	1.4	20	872	145	0.166
camel	1.6	20	965	188	0.195
jedit	3.2	20	272	90	0.331
jedit	4.0	20	306	75	0.245
jedit	4.1	20	312	79	0.253
jedit	4.2	20	367	48	0.131
jedit	4.3	20	492	11	0.022
log4j	1.0	20	135	34	0.252
log4j	1.1	20	109	37	0.339
log4j	1.2	20	205	189	0.922
xalan	2.4	20	723	110	0.152
xalan	2.7	20	909	898	0.988

TABLE 5. Average AUC results obtained by BMFO-S2 with different number of search agents [KNN as the base classifier].

Dataset	No. of search agents					
	5	10	20	30	40	50
ant-1.7	0.7048	**0.7060**	0.6963	0.7027	0.7037	0.6992
camel-1.0	0.4994	0.4997	0.4985	**0.5057**	0.4986	0.4988
camel-1.2	0.5866	0.5819	**0.5887**	0.5861	0.5744	0.5793
camel-1.4	0.5852	**0.5969**	0.5924	0.5948	0.5943	0.5963
camel-1.6	0.5620	0.5621	0.5697	0.5692	**0.5762**	0.5760
jedit-3.2	0.7440	0.7443	0.7362	0.7443	**0.7506**	0.7412
jedit-4.0	0.6867	0.6959	0.6928	0.6952	0.6985	**0.6991**
jedit-4.1	0.7186	0.7202	**0.7279**	0.7238	0.7179	0.7185
jedit-4.2	0.6558	0.6660	0.6655	0.6573	**0.6688**	0.6648
jedit-4.3	0.5000	0.5000	0.5070	**0.5291**	0.5285	0.5140
log4j-1.0	0.7757	0.7718	**0.7851**	0.7851	0.7817	0.7841
log4j-1.1	0.7964	0.7964	**0.8146**	0.8058	0.8051	0.7983
log4j-1.2	0.6517	0.6819	0.6699	0.6738	0.6770	**0.6881**
xalan-2.4	0.6094	**0.6138**	0.6108	0.6119	0.6075	0.6102
xalan-2.7	0.8268	0.8269	0.8494	0.8449	**0.8496**	0.8450
Rank (F_Test)	4.767	3.300	3.467	**2.867**	3.200	3.400

Operating Characteristic curve- area under curve (AUC). Accuracy and precision are easily affected by cut-off value once a little change in the dataset classes, while AUC is not affected by the cut-off value. In this work, we evaluate the proposed approach based on AUC value.

The calculation of AUC value depends on the ratio between True Positive (TP) rate verse False Positive (FP) rate. A confusion matrix as shown in Table 2 define the process of evaluating AUC value, where TP (True positive) when both actual and predicted value are correct positive. FP (False Positive) when both actual and predicted value are correct negative. FN (False Negative) when predicted value is negative, while actual value is positive. TN (True Negative) when predicted value is positive, while actual value is negative.

The AUC value depends on two values: sensitivity (Equation 10) and specificity (Equation 10), where

TABLE 6. Average AUC values obtained by BMFO-S2 with ADASYN using different balancing ratios [KNN as the base classifier].

Dataset	original	ADASYN balance_ratio			
		0.4	0.6	0.8	1
ant-1.7	0.702675	0.73594	0.736492	0.724549	**0.737576**
camel-1.0	0.505698	0.668546	0.691836	0.704637	**0.714535**
camel-1.2	0.586087	0.58424	0.586413	**0.60043**	0.597482
camel-1.4	0.594824	0.624855	0.633772	0.637803	**0.640053**
camel-1.6	0.569219	0.616196	0.626651	**0.641625**	0.637068
jedit-3.2	0.744328	**0.746893**	0.743413	0.739347	0.73917
jedit-4.0	0.695203	**0.708926**	0.703022	0.705801	0.707931
jedit-4.1	0.723793	0.735454	**0.744934**	0.739192	0.741848
jedit-4.2	0.657305	0.743659	**0.759816**	0.759796	0.759581
jedit-4.3	0.529087	0.701862	0.693933	**0.70824**	0.692941
log4j-1.0	**0.785061**	0.744758	0.748966	0.742938	0.735032
log4j-1.1	0.805781	0.801408	**0.815146**	0.802571	0.801032
log4j-1.2	0.673793	**0.736491**	0.730688	0.735946	0.720866
xalan-2.4	0.611929	0.692821	0.711697	0.722529	**0.729023**
xalan-2.7	0.844898	**0.889365**	0.860569	0.868086	0.878123
Rank (F_Test)	4.27	3.00	2.60	**2.47**	2.67

TABLE 7. A comparison between the performance of BMFO-S2 in dealing with imbalanced versus balanced data in terms of evaluation metrics [KNN as the base classifier].

Dataset	Sensitivity		Specificity		AUC	
	original	ADASYN	original	ADASYN	original	ADASYN
ant-1.7	0.5094	**0.7456**	**0.8960**	0.7035	0.7027	**0.7245**
camel-1.0	0.0154	**0.6231**	**0.9960**	0.7862	0.5057	**0.7046**
camel-1.2	0.3727	**0.5491**	**0.7995**	0.6518	0.5861	**0.6004**
camel-1.4	0.2372	**0.5828**	**0.9524**	0.6928	0.5948	**0.6378**
camel-1.6	0.2027	**0.6234**	**0.9358**	0.6598	0.5692	**0.6416**
jedit-3.2	0.6678	**0.7611**	**0.8209**	0.7176	**0.7443**	0.7393
jedit-4.0	0.4973	**0.7133**	**0.8931**	0.6983	0.6952	**0.7058**
jedit-4.1	0.5532	**0.7342**	**0.8944**	0.7442	0.7238	**0.7392**
jedit-4.2	0.3729	**0.7500**	**0.9417**	0.7696	0.6573	**0.7598**
jedit-4.3	0.0727	**0.5909**	**0.9854**	0.8256	0.5291	**0.7082**
log4j-1.0	0.6176	**0.7265**	**0.9525**	0.7594	**0.7851**	0.7429
log4j-1.1	0.6838	**0.7649**	**0.9278**	0.8403	**0.8058**	0.8026
log4j-1.2	**0.9788**	0.7344	0.3688	**0.7375**	0.6738	**0.7359**
xalan-2.4	0.2718	**0.7464**	**0.9520**	0.6987	0.6119	**0.7225**
xalan-2.7	**0.9989**	0.9816	0.6909	**0.7545**	0.8449	**0.8681**
W|L	2|13	13|2	13|2	2|13	3|12	12|3

TABLE 8. A comparative results of BMFO-S2 using different classifiers with ADASYN oversampling in terms of sensitivity, specificity, AUC, and running time.

Dataset	Sensetivity			Specificity			AUC			Time		
	KNN	DT	LDA	KNN	DT	LDA	KNN	DT	LDA	KNN	DT	LDA
ant-1.7	**0.7456**	0.5596	0.6591	0.7035	0.7948	**0.8420**	0.7245	0.6772	**0.7505**	**872.02**	1666.26	1151.78
camel-1.0	0.6231	0.3308	**0.6308**	0.7862	**0.9347**	0.8454	0.7046	0.6327	**0.7381**	**491.71**	771.81	722.91
camel-1.2	**0.5491**	0.4912	0.5269	0.6518	0.7153	**0.7291**	0.6004	0.6033	**0.6280**	**853.95**	1538.06	1126.07
camel-1.4	**0.5828**	0.3703	0.5469	0.6928	**0.8367**	0.8110	0.6378	0.6035	**0.6790**	**908.79**	1842.83	1141.52
camel-1.6	**0.6234**	0.3649	0.5569	0.6598	**0.8257**	0.7456	0.6416	0.5953	**0.6512**	**1057.66**	2077.39	1307.92
jedit-3.2	0.7611	0.6033	**0.7989**	0.7176	0.7764	**0.8176**	0.7393	0.6899	**0.8082**	**541.17**	772.78	767.32
jedit-4.0	0.7133	0.5827	**0.7547**	0.6983	**0.8238**	0.6900	0.7058	0.7032	**0.7224**	**552.27**	823.26	773.53
jedit-4.1	**0.7342**	0.5835	0.7101	0.7442	0.8009	**0.8322**	0.7392	0.6922	**0.7712**	**559.90**	831.96	776.29
jedit-4.2	**0.7500**	0.4813	0.7458	0.7696	**0.8649**	0.8260	0.7598	0.6731	**0.7859**	**545.62**	898.40	769.64
jedit-4.3	**0.5909**	0.4091	0.4182	0.8256	**0.9480**	0.8669	**0.7082**	0.6786	0.6426	**550.08**	910.01	757.27
log4j-1.0	0.7265	0.6412	**0.7324**	0.7594	0.7911	**0.8248**	0.7429	0.7161	**0.7786**	**466.45**	587.19	675.37
log4j-1.1	**0.7649**	0.7027	0.7622	**0.8403**	0.7847	0.8111	**0.8026**	0.7437	0.7866	**456.95**	541.36	664.01
log4j-1.2	0.7344	**0.9360**	0.7561	**0.7375**	0.4875	0.5938	**0.7359**	0.7117	0.6749	**472.62**	620.58	685.14
xalan-2.4	**0.7464**	0.4291	0.6500	0.6987	**0.8315**	0.7886	**0.7225**	0.6303	0.7193	**777.26**	1561.95	1009.29
xalan-2.7	0.9816	**0.9943**	0.7664	0.7545	0.6727	**0.9636**	**0.8681**	0.8335	0.8650	**700.28**	908.29	869.79
W : L	9: 6	2:13	4:11	2:13	7: 8	6: 9	5:10	0:15	10: 5	15:0	0:15	0:15
Rank	**1.47**	2.73	1.8	2.6	1.73	**1.67**	1.73	2.8	**1.47**	**1**	2.8	2.2

TABLE 9. Comparative results of LDA classifier before and after feature selection [based on re-sampled data and BMFO-S2 as FS].

Dataset	Sensetivity		Specificity		AUC		#features		Time	
	without	with	without	with	without	with	without	with	without	with
ant-1.7	0.6491	**0.6591**	0.8031	**0.8420**	0.7261	**0.7505**	20	**10**	**1.4504**	1151.78
camel-1.0	0.3846	**0.6308**	0.8282	**0.8454**	0.6064	**0.7381**	20	**10**	**0.2704**	722.91
camel-1.2	0.5185	**0.5269**	0.7168	**0.7291**	0.6177	**0.6280**	20	**13.8**	**0.1875**	1126.07
camel-1.4	0.5241	**0.5469**	**0.8239**	0.8110	0.6740	**0.6790**	20	**12.9**	**0.1458**	1141.52
camel-1.6	0.5479	**0.5569**	**0.7465**	0.7456	0.6472	**0.6512**	20	**14.9**	**0.1829**	1307.92
jedit-3.2	0.7444	**0.7989**	0.7912	**0.8176**	0.7678	**0.8082**	20	**15.4**	**0.1010**	767.32
jedit-4.0	0.6667	**0.7547**	**0.6926**	0.6900	0.6797	**0.7224**	20	**10.7**	**0.1084**	773.53
jedit-4.1	0.7089	**0.7101**	**0.8326**	0.8322	0.7707	**0.7712**	20	**13.2**	**0.1064**	776.29
jedit-4.2	**0.7500**	0.7458	0.8245	**0.8260**	**0.7872**	0.7859	20	**11.9**	**0.1096**	769.64
jedit-4.3	0.3636	**0.4182**	0.7942	**0.8669**	0.5789	**0.6426**	20	**10**	**0.1175**	757.27
log4j-1.0	0.5882	**0.7324**	0.7822	**0.8248**	0.6852	**0.7786**	20	**10.3**	**0.1161**	675.37
log4j-1.1	0.6486	**0.7622**	0.7639	**0.8111**	0.7063	**0.7866**	20	**12.7**	**0.1069**	664.01
log4j-1.2	**0.7619**	0.7561	0.3750	**0.5938**	0.5685	**0.6749**	20	**11.5**	**0.1203**	685.14
xalan-2.4	0.5909	**0.6500**	0.7765	**0.7886**	0.6837	**0.7193**	20	**10.9**	**0.1282**	1009.29
xalan-2.7	**0.7817**	0.7664	0.7273	**0.9636**	0.7545	**0.8650**	20	**9.9**	**0.1405**	869.79

P represents the number of actually positive records and N is the number of actually negative records. Table 3 demonstrates a description of AUC values scale.

$$Sensitivity = TP_{rate} = \frac{TP}{P} \tag{9}$$

$$Specificity = TN_{rate} = \frac{TN}{N} \tag{10}$$

B. FEATURE SELECTION FITNESS FUNCTION

Since FS problem is an optimization problem, the fitness function that used in this work to evaluate the selected features is presented in Equation 11. Where α is a float number between 0 and 1 that is selected randomly, $\lambda_R(D)$ is the error rate for internal classifier, β is a float number selected randomly between 0 and α, $|R|$ represents a number of selected features, and finally $|N|$ represents the overall number of features.

$$Fitness = min[\alpha\lambda_R(D) + \beta\frac{|R|}{|N|}] \tag{11}$$

VIII. EXPERIMENTAL RESULTS AND SIMULATIONS

To evaluate the proposed approach, 15 different software fault projects obtained from PROMISE public software engineering repository are selected [52], [53]. The datasets have no missing data and well structured for research use. Table 4 demonstrates PROMISE datasets. The selected dataset is an object-oriented metrics, where each project consists of 20 features (input metrics) and a one binary output as a fault value. Table 17 in Appendix IX shows a description of each feature in the datasets used in this paper.

At the beginning, we examine the performance of BMFO-S2 using different number of agents (moths). Table 5 shows the average AUC values using different size of agents (i.e 5, 10, 20, 30, 40, and 50) using kNN classifier. It is clear that number of agents play a vital role of the overall performance. For example, the performance is worst when number of agent is small (i.e Size = 5), while the best performance is achieved when agent size is 30. Choosing the correct size of agents will play a vital role on the final results.

TABLE 10. Comparison between different variants of BMFO based on S-shaped TFs in terms of number of features, AUC, and fitness results. [based on LDA classieir with ADASYN].

Dataset	# selected features				AUC				Fitness			
	BMFOS1	BMFOS2	BMFOS3	BMFOS4	BMFOS1	BMFOS2	BMFOS3	BMFOS4	BMFOS1	BMFOS2	BMFOS3	BMFOS4
ant-1.7	13.3	**10**	10.5	11.2	0.7480	**0.7505**	0.7496	0.7476	0.2561	**0.2520**	0.2532	0.2555
camel-1.0	11.5	10	9	**8.5**	0.6979	**0.7381**	0.7299	0.7124	0.3048	**0.2643**	0.2719	0.2890
camel-1.2	14.3	13.8	13.1	**12.8**	0.6278	**0.6280**	0.6197	0.6261	0.3757	**0.3752**	0.3831	0.3765
camel-1.4	14.5	12.9	**12**	12.3	0.6782	0.6790	**0.6840**	0.6791	0.3258	0.3243	**0.3188**	0.3239
camel-1.6	15.9	14.9	14.1	**13**	**0.6550**	0.6512	0.6480	0.6492	**0.3495**	0.3527	0.3556	0.3538
jedit-3.2	15.1	15.4	13.3	**12.1**	0.8077	**0.8082**	0.8041	0.8013	0.1979	**0.1975**	0.2006	0.2027
jedit-4.0	12.8	10.7	10.3	**10.1**	0.7217	**0.7224**	0.7213	0.7223	0.2819	0.2802	0.2811	**0.2800**
jedit-4.1	14.5	13.2	11.5	**11.4**	0.7772	0.7712	**0.7797**	0.7705	0.2278	0.2332	**0.2238**	0.2329
jedit-4.2	13.7	11.9	11.5	**10.4**	0.7829	0.7859	0.7829	**0.7944**	0.2218	0.2179	0.2207	**0.2087**
jedit-4.3	14.2	**10**	11.1	11.1	0.6423	0.6426	**0.6766**	0.6708	0.3612	0.3589	**0.3258**	0.3314
log4j-1.0	11.7	10.3	**9.2**	**9.2**	0.7575	**0.7786**	0.7706	0.7717	0.2460	**0.2244**	0.2317	0.2307
log4j-1.1	11.9	12.7	**9.6**	11	**0.7873**	0.7866	0.7780	0.7763	**0.2165**	0.2176	0.2246	0.2270
log4j-1.2	13.2	11.5	10.4	**10.2**	0.6904	0.6749	0.6727	**0.7177**	0.3131	0.3276	0.3292	**0.2846**
xalan-2.4	14.2	10.9	10.7	**10.2**	0.7177	0.7193	**0.7251**	0.7230	0.2866	0.2834	**0.2775**	0.2793
xalan-2.7	11.7	9.9	8.8	**7.8**	0.8702	0.8650	**0.8792**	0.8702	0.1344	0.1386	**0.1240**	0.1324
W|T|L	0|0|15	2|0|13	2|1|12	10|1|4	2|0|13	6|0|9	5|0|10	2|0|13	2|0|13	5|0|10	5|0|10	3|0|12
Rank (Ftest)	3.87	2.87	1.87	**1.4**	2.8	**2.07**	2.5	2.63	3	**2.2**	2.4	2.4

TABLE 11. Comparison between different variants of BMFO based on V-shaped TFs in terms of number of features, AUC, and fitness results. [based on LDA classieir with ADASYN].

Dataset	# selected features				AUC				Fitness			
	BMFOV1	BMFOV2	BMFOV3	BMFOV4	BMFOV1	BMFOV2	BMFOV3	BMFOV4	BMFOV1	BMFOV2	BMFOV3	BMFOV4
ant-1.7	**6.80**	6.90	**6.80**	7.10	0.7551	0.7543	**0.7553**	0.7536	0.2459	0.2467	**0.2456**	0.2475
camel-1.0	**4.70**	5.50	4.80	5.30	0.7670	0.7564	0.7404	**0.7755**	0.2330	0.2439	0.2594	**0.2249**
camel-1.2	**8.20**	9.00	10.10	10.10	0.6245	**0.6260**	0.6220	0.6158	0.3759	**0.3748**	0.3793	0.3854
camel-1.4	10.70	**9.70**	10.80	10.60	0.6789	0.6848	**0.6889**	0.6858	0.3233	0.3169	**0.3134**	0.3163
camel-1.6	**9.20**	10.40	10.60	12.10	0.6550	0.6484	0.6545	**0.6582**	0.3462	0.3532	0.3473	**0.3444**
jedit-3.2	**8.90**	9.40	9.20	10.00	0.7944	0.7985	0.8008	**0.8049**	0.2080	0.2042	0.2019	**0.1981**
jedit-4.0	5.80	5.80	6.20	**5.70**	0.7370	0.7354	0.7362	**0.7412**	0.2632	0.2649	0.2642	**0.2590**
jedit-4.1	**6.70**	7.10	7.10	7.40	0.7753	**0.7885**	0.7806	0.7860	0.2258	**0.2129**	0.2208	0.2155
jedit-4.2	7.50	**6.50**	7.10	8.20	0.7981	0.7991	**0.8042**	0.7915	0.2037	0.2021	**0.1974**	0.2105
jedit-4.3	4.60	5.60	3.60	**2.90**	**0.7229**	0.7095	0.7185	0.7188	**0.2767**	0.2904	0.2805	0.2798
log4j-1.0	**4.90**	**4.90**	6.00	5.60	**0.7904**	0.7890	0.7815	0.7854	**0.2099**	0.2114	0.2193	0.2153
log4j-1.1	2.40	2.00	**1.90**	2.60	0.8028	**0.8108**	0.8081	0.8060	0.1964	**0.1883**	0.1909	0.1933
log4j-1.2	5.80	6.40	**5.50**	7.00	0.7107	0.6824	**0.7173**	0.7017	0.2893	0.3176	**0.2826**	0.2988
xalan-2.4	7.00	6.40	6.50	**6.30**	0.7271	**0.7348**	0.7340	0.7322	0.2736	**0.2657**	0.2666	0.2683
xalan-2.7	3.50	3.20	3.40	**2.90**	0.8676	**0.8780**	0.8661	0.8643	0.1328	**0.1223**	0.1342	0.1358
W|T|L	5|2|8	2|1|12	2|1|12	4|0|11	2|0|13	5|0|10	4|0|11	4|0|11	2|0|13	5|0|10	4|0|11	4|0|11
Rank (Ftest)	**2.17**	2.37	2.57	2.9	2.6	2.47	**2.4**	2.53	2.6	2.47	**2.4**	2.53

TABLE 12. Comparison between the top variants of BMFO with S-shaped and V-shaped versus the corresponding proposed methods based on the number of features, AUC, and fitness results.

Dataset	# selected features				AUC				Fitness			
	S-shaped		V-shaped		S-shaped		V-shaped		S-shaped		V-shaped	
	BMFOS2	EBMFOS2	BMFOV3	EBMFOV3	BMFOS2	EBMFOS2	BMFOV3	EBMFOV3	BMFOS2	EBMFOS2	BMFOV3	EBMFOV3
ant-1.7	**10.00**	10.3	6.80	**6.2**	0.7505	**0.7552**	0.7553	**0.7615**	0.2520	**0.2475**	0.2456	**0.2392**
camel-1.0	10.00	**8.7**	**4.80**	5.3	0.7381	**0.7489**	0.7404	**0.7552**	0.2643	**0.2529**	0.2594	**0.2450**
camel-1.2	13.80	**13.4**	**10.10**	10.8	**0.6280**	0.6217	**0.6220**	0.6215	**0.3752**	0.3813	**0.3793**	0.3801
camel-1.4	12.90	**12.7**	10.80	**8.2**	0.6790	**0.6845**	0.6889	**0.6918**	0.3243	**0.3187**	0.3134	**0.3092**
camel-1.6	14.90	**14.3**	10.60	**10.3**	0.6512	**0.6727**	0.6545	**0.6580**	0.3527	**0.3312**	0.3473	**0.3437**
jedit-3.2	15.40	**12.5**	**9.20**	10.1	**0.8082**	0.8028	0.8008	**0.8053**	**0.1975**	0.2015	0.2019	**0.1978**
jedit-4.0	**10.70**	11.5	**6.20**	7.8	0.7224	**0.7253**	**0.7362**	0.7161	0.2802	**0.2777**	**0.2642**	0.2850
jedit-4.1	13.20	**12**	**7.10**	7.7	0.7712	**0.7814**	0.7806	**0.7868**	0.2332	**0.2225**	0.2208	**0.2149**
jedit-4.2	11.90	**11.4**	**7.10**	7.8	0.7859	**0.7937**	**0.8042**	0.7973	0.2179	**0.2100**	**0.1974**	0.2046
jedit-4.3	**10.00**	10.6	**3.60**	6.1	0.6426	**0.7028**	0.7185	**0.7499**	0.3589	**0.2995**	0.2805	**0.2506**
log4j-1.0	10.30	**9.8**	6.00	**5.3**	**0.7786**	0.7682	0.7815	**0.7937**	**0.2244**	0.2344	0.2193	**0.2069**
log4j-1.1	12.70	**10.9**	**1.90**	2.5	**0.7866**	0.7659	**0.8081**	0.7986	**0.2176**	0.2372	**0.1909**	0.2006
log4j-1.2	11.50	**10.4**	5.50	**5.2**	0.6749	**0.6910**	**0.7173**	0.7102	0.3276	**0.3111**	**0.2826**	0.2895
xalan-2.4	10.90	**10.6**	6.50	**5.6**	0.7193	**0.7266**	0.7340	**0.7495**	0.2834	**0.2759**	0.2666	**0.2507**
xalan-2.7	**9.90**	12.5	**3.40**	8.2	**0.8650**	0.8263	**0.8661**	0.8456	**0.1386**	0.1782	**0.1342**	0.1570
W|T|L	4|0|11	1|1|0|4	9|0|6	6|0|9	5|0|10	10|0|5	6|0|9	9|0|6	5|0|10	10|0|5	6|0|9	9|0|6
Overall rank	3.73	3.27	**1.4**	1.6	3.27	2.87	**2**	**1.87**	3.27	3	1.93	**1.8**

Table 6 explores the obtained results of original BMFO-S2 and modified BMFO-S2 with ADASYN using kNN classifiers. We employed different balancing ratio (i.e. 0.4, 0.6, 0.8 and 1.0). It is clear that the modified BMFO-S2 with ADASYN outperforms the original BMFO-S2 algorithm. Moreover, balancing ratio affects the performance of

TABLE 13. Comparison between EBMFOS2, EBMFOV3 and other metaheuristics in terms of average AUC, and average number of features [based on LDA classifier with ADASYN 0.8].

Dataset	AUC							#selected features						
	EBMFOS2	EBMFOV3	BGOA	BGSA	WOA	BBA	BALO	EBMFOS2	EBMFOV3	BGOA	BGSA	WOA	BBA	BALO
ant-1.7	0.7552	**0.7615**	0.7534	0.7453	0.7531	0.7354	0.7504	10.3	**6.2**	10.9	11.4	11.1	9.3	14.8
camel-1.0	0.7489	**0.7552**	0.7230	0.7041	0.7461	0.6681	0.6896	8.7	**5.3**	8.3	8.5	9.4	7.5	14.0
camel-1.2	0.6217	0.6215	**0.6240**	0.6113	0.6239	0.5831	0.6146	13.4	10.8	11.4	11.7	13.7	**9.0**	16.3
camel-1.4	0.6845	**0.6918**	0.6783	0.6725	0.6829	0.6602	0.6751	12.7	**8.2**	13.6	11.6	14.3	10.3	16.5
camel-1.6	**0.6727**	0.6580	0.6607	0.6466	0.6544	0.6175	0.6542	14.3	**10.3**	14.8	10.5	15.5	10.6	17.5
jedit-3.2	0.8028	**0.8053**	0.8032	0.7933	0.7983	0.7825	0.7972	12.5	**10.1**	12.7	13.2	13.5	11.7	17.6
jedit-4.0	**0.7253**	0.7161	0.7176	0.7148	0.7186	0.7061	0.6968	11.5	**7.8**	10.0	9.1	11.2	8.2	16.5
jedit-4.1	0.7814	**0.7868**	0.7787	0.7648	0.7812	0.7500	0.7746	12.0	**7.7**	11.4	11.1	12.6	10.6	18.5
jedit-4.2	0.7937	**0.7973**	0.7862	0.7771	0.7867	0.7650	0.7683	11.4	**7.8**	11.4	11.2	11.6	10.2	15.4
jedit-4.3	0.7028	**0.7499**	0.6927	0.6280	0.6392	0.5922	0.6439	10.6	**6.1**	10.1	11.9	11.0	9.5	15.8
log4j-1.0	0.7682	**0.7937**	0.7815	0.7663	0.7618	0.7314	0.7560	9.8	**5.3**	7.6	9.2	9.0	9.6	14.3
log4j-1.1	0.7659	**0.7986**	0.7820	0.7711	0.7745	0.7425	0.7528	10.9	**2.5**	10.4	8.6	12.0	8.3	16.1
log4j-1.2	0.6910	0.7102	0.7074	0.6636	**0.7137**	0.6517	0.6469	10.4	**5.2**	9.8	9.9	11.2	8.5	15.1
xalan-2.4	0.7266	**0.7495**	0.7260	0.7013	0.7247	0.6915	0.7107	10.6	**5.6**	11.6	10.0	9.6	8.8	16.1
xalan-2.7	0.8263	0.8456	**0.8699**	0.8378	0.8697	0.8165	0.8526	12.5	8.2	**6.8**	9.7	8.2	9.5	13.5
W|L	2|13	10|5	2|13	0|15	1|14	0|15	0|15	0|15	13|2	1|14	0|15	0|15	1|14	0|15
Rank (Ftest)	2.67	**1.8**	2.73	5.4	3.2	6.87	5.33	4.7	**1.17**	3.7	3.93	5.17	2.33	7

BMFO-S2 algorithm. For example, balancing ratio equals 0.8 shows a good performance compared to other ratios. Table 7 shows a comparison between original BMFO-S2 and modified BMFO-S2 with ADASYN over balanced and imbalanced datasets. It is clear that the performance of KNN classifier is outstanding based on the AUC value for balanced datasets. The reported results in Tables 6 and 7 show the balance datasets will enhance the performance of kNN classifiers compared to imbalanced one.

Table 8 explores the obtained results for modified BMFO-S2 using three different classifiers (ie. kNN, DT and LDA). The performance of DT is the worst based on AUC values, while the performance of LDA is best. However, the execution time for kNN is better than DT and LDA. In general, Software fault prediction problem works offline. So, if we are looking for outstanding classifiers, LDA will be the first choice since project developers have time to predict and evaluate their new products.

For more investigation about the activeness of the proposed approach, we executed our approach using two strategies, with and without FS. We believe that will give us a big picture about the performance of the classifiers while using FS or not. Table 9 shows the obtained results before and after FS with re-sampled data. It is clear that the AUC values are improved with FS, while the computational time is increased. In reality, execution time for software fault prediction is not an important factor due to all experiments are done offline.

Since LDA classifiers outperforms all other classifiers, Table 10 compares 4 different versions of BMFO based on S-shaped TFs (i.e. BMFOS1, BMFOS2, BMFOS3 and BMFOS4) using LDA classifiers with ADASYN. The experiments show that executing feature selection will have a high impact on the overall performance. Based on AUC and fitness values, BMFOS2 gain the first rank. Table 11 reports the performance of BMFO based on V-shape TFs (i.e. BMFOV1, BMFOV2, BMFOV3 and BMFOV4). The obtained results show that BMFOV3 outperforms all other versions base on rank.

TABLE 14. The total number of selections by EBMFOV3 algorithm for each feature.

Sequence #	Feature	Number of selections
5	rfc	77
12	dam	74
1	wmc	68
4	cbo	58
9	npm	57
14	mfa	54
2	dit	53
6	lcom	51
20	avg_cc	49
11	loc	48
7	ca	47
10	lcom3	46
18	amc	46
8	ce	42
17	cbm	42
19	max_cc	36
3	noc	35
15	cam	32
16	ic	31
13	moa	28

From Tables 10 and 11, we found that BMFOS2 (from S-shape group) and BMFOV3 (from V-shape group) show an excellent performance compared to other versions. As a result, we employed both versions with BMFO and EBMFO as shown in Table 12. The obtained results show that EBMFO for both groups (S-shape and V-shape) shows a stable and robust performance compared to BMFO.

The comparisons with the most related meta-heuristic algorithms were conducted in two phases. In the first one, all algorithms were implemented and executed in the same environment and using the same parameter settings to make fair comparisons. In the second phase, the results of well-known approaches that use the same datasets were obtained from the literature, and an in-depth comparison was conducted. All comparisons were held based on the AUC matrix and the number of selected features. Table 13 shows the comparisons between the best performing MFO based approaches

TABLE 15. The selected features by EBMFOV3 for the best AUC results for each dataset.

Dataset	No. features	Features										Best AUC
ant-1.7	5	cbo	rfc	lcom3	dam	mfa						0.763025
camel-1.0	4	wmc	cbo	lcom	cbm							0.795658
camel-1.2	10	wmc	noc	rfc	lcom	npm	loc	dam	moa	mfa	avg_cc	0.631708
camel-1.4	7	cbo	ca	ce	npm	mfa	cam	max_cc				0.70785
camel-1.6	10	wmc	dit	noc	cbo	ca	ce	npm	lcom3	cbm	avg_cc	0.663767
jedit-3.2	7	wmc	dit	rfc	lcom3	loc	dam	avg_cc				0.803785
jedit-4.0	6	wmc	cbo	rfc	ca	lcom3	dam					0.757143
jedit-4.1	7	rfc	loc	dam	mfa	ic	cbm	avg_cc				0.802276
jedit-4.2	5	rfc	lcom	cam	max_cc	avg_cc						0.815896
jedit-4.3	4	cbo	ca	ce	amc							0.728029
log4j-1.0	5	wmc	dit	noc	lcom	avg_cc						0.822656
log4j-1.1	2	loc	amc									0.823198
log4j-1.1	2	loc	amc									0.823198
log4j-1.2	5	dit	npm	dam	mfa	amc						0.774471
xalan-2.4	7	wmc	rfc	lcom	mfa	ic	cbm	amc				0.743616
xalan-2.7	4	wmc	cbo	npm	amc							0.896993

TABLE 16. Comparison between the proposed approach and other methods in the literature in terms of AUC.

Dataset	Our results		Shatnawi [24]				Okutan and Yildiz [55]	Turabieh and Mafarja [1]
	EBMFOS2	EBMFOV3	LR	NB	5NN	C4.5	Bayesian networks	L-RNN without CV
ant-1.7	0.7552	0.7615	**0.830**	0.790	0.760	0.740	0.820	0.523
camel-1.0	0.7489	**0.7552**	-	-	-	-	-	0.607
camel-1.2	0.6217	0.6215	0.570	0.560	**0.640**	0.520	-	0.443
camel-1.4	0.6845	0.6918	**0.700**	0.670	0.670	0.600	-	0.521
camel-1.6	**0.6727**	0.6580	0.650	0.590	0.660	0.540	-	0.505
jedit-3.2	0.8028	**0.8053**	-	-	-	-	-	0.518
jedit-4.0	0.7253	0.7161	0.770	0.700	**0.810**	0.720	-	0.549
jedit-4.1	0.7814	0.7868	**0.820**	0.750	0.800	0.690	-	-
jedit-4.2	0.7937	0.7973	**0.840**	0.750	0.770	0.640	-	0.519
jedit-4.3	0.7028	**0.7499**	-	-	-	-	0.658	0.684
log4j-1.0	0.7682	**0.7937**	-	-	-	-	-	0.535
log4j-1.1	0.7659	**0.7986**	-	-	-	-	-	0.522
log4j-1.2	0.6910	**0.7102**	-	-	-	-	-	0.361
xalan-2.4	0.7266	**0.7495**	-	-	-	-	-	0.536
xalan-2.7	0.8263	**0.8456**	-	-	-	-	-	0.420

and the implemented meta-heuristic algorithms (i.e., BGOA, BGSA, WOA, BBA, and BALO). Based on AUC values, it is clear that EBMFOV3 obtained the best results among all algorithms in 67% of the datasets. BGOA obtained the best results in 13% of the datasets while WOA outperformed other algorithms in one dataset only. This proves the efficiency of the proposed approach in selecting the most relevant features that significantly enhance the performance of the MFO algorithm. The same behavior of the EBMFOV3 when the number of selected features is investigated. EBMFOV3 obtained the minimum of selected features in 87% of the datasets. While BGOA and BBA could select the minimum number of features in one dataset only. The obtained results prove the ability of the proposed approach to explore the search space efficiently and find the best performing solutions. We believe that updating the population using the EPD concept helped the algorithm to escape from the local optima by increasing the population diversity. This can be seen obviously when observing the reported results in the aforementioned tables.

Table 14 shows the frequency of selecting each feature from all datasets using the EBMFOV3 FS approach.The presented numbers in this table were obtained after 10 runs

TABLE 17. The description of features.

Feature	Description
wmc	Number of methods defined in a class
dit	Depth of a class within the class hierarchy from the root of inheritance.
noc	Number of immediate descendants of a class.
cbo	Count the number of classes coupled to class.
rfc	Count the number of distinct methods invoked by a class in response to a received message.
lcom	Count the number of methods that do not share a field to the method pairs that do.
ca	Count the number of dependent classes for a given class.
ce	Count the number of classes on which a class depends.
npm	Number of public methods defined in a class.
lcom3	Count the number of connected components in a method graph.
loc	Count the total number of lines of code of a class.
dam	Computes the ratio of private attributes in a class.
moa	Count the number of data members declared as class type.
mfa	Shows the fraction of the methods inherited by a class to the methods accessible by the functions defined in the class.
cam	Computes the cohesion among methods of a class based on the parameters list.
ic	Count the number of coupled ancestor classes of a class.
cbm	Count the number of new or redefined methods that are coupled with the inherited methods.
amc	Measures the average method size for each class.
max_cc	Maximum counts of the number of logically independent paths in a method.
avg_cc	Average counts of the number of logically independent paths in a method.

for each dataset. From Table 14, it can be seen that rfc feature has been selected for 77 times, which indicates that importance of this feature in the prediction process. The same observation can be made when observing features dam and wmc, that were selected for 74 and 68 times respectively. As a conclusion, to generate a good classification model for SFP problem, the most frequent features (e.g., rfc, dam and wmc) metric should not be ignored during collecting data for any new project. The selected features in the solution with the best AUC results, for each dataset, were reported in Table 15. Table 16 shows a comparison between the proposed approach and other methods in the literature in terms of AUC value. It is clear that the proposed approach is outperform the reported results in eight datasets using EBMFOV3 and one dataset using EBMFOS2 compared to other methods.

We are interested to compare the obtained (AUC) results from the proposed approaches with the results of a set of SFP approaches that used feature selection algorithms or fixed size of features.

IX. CONCLUSION AND FUTURE WORKS

In this paper, an intelligent approach to predict software fault based on a Binary Moth Flame Optimization (BMFO) with Adaptive synthetic sampling (ADASYN) was introduced. BMFO works as a wrapper feature selection, while ADASYN works to overcome imbalanced data problem. Several classifiers such as kNN, DT, and LDA are used to predict software faults. The proposed approach improves the performance of all classifiers after solving imbalanced problems. LDA outperforms other classifiers based on AUC value, while kNN execution time is the best. The performance of EBFMO with V-shape (TFs = 3) outperforms other versions and all results in the literature. It is clear that from the obtained results, the importance of feature selection algorithm for classification problems and build an accurate system to predict software faults. In the future work, we will study the importance of features to enhance the performance of classifiers and SFP model accuracy.

APPENDIXES

See the table 17.

REFERENCES

[1] H. Turabieh, M. Mafarja, and X. Li, "Iterated feature selection algorithms with layered recurrent neural network for software fault prediction," *Expert Syst. Appl.*, vol. 122, pp. 27–42, May 2019.

[2] A. Porter and R. Selby, "Empirically guided software development using metric-based classification trees," *IEEE Softw.*, vol. 7, no. 2, pp. 46–54, Mar. 1990.

[3] R. Hoda, N. Salleh, J. Grundy, and H. M. Tee, "Systematic literature reviews in agile software development: A tertiary study," *Inf. Softw. Technol.*, vol. 85, pp. 60–70, May 2017.

[4] A. M. Johnson and M. Malek, "Survey of software tools for evaluating reliability, availability, and serviceability," *ACM Comput. Surv.*, vol. 20, no. 4, pp. 227–269, Sep. 1988, doi: 10.1145/50020.50062.

[5] Y. Xia, G. Yan, and Q. Si, "A study on the significance of software metrics in defect prediction," in *Proc. 6th Int. Symp. Comput. Intell. Design*, vol. 2, Oct. 2013, pp. 343–346.

[6] S. S. Rathore and S. Kumar, "A decision tree logic based recommendation system to select software fault prediction techniques," *Computing*, vol. 99, no. 3, pp. 255–285, Mar. 2017.

[7] T. Zimmermann, N. Nagappan, and A. Zeller, "Predicting bugs from history," *Softw. Evol.*, vol. 4, no. 1, pp. 69–88, 2008.

[8] H. Liu and H. Motoda, *Feature Selection for Knowledge Discovery and Data Mining*, vol. 454. New York, NY, USA: Springer, 2012.

[9] T. W. Rauber, F. De Assis Boldt, and F. M. Varejao, "Heterogeneous feature models and feature selection applied to bearing fault diagnosis," *IEEE Trans. Ind. Electron.*, vol. 62, no. 1, pp. 637–646, Jan. 2015.

[10] M. Dash and H. Liu, "Feature selection for classification," *Intell. Data Anal.*, vol. 1, nos. 1–4, pp. 131–156, 1997.

[11] T. Hall, S. Beecham, D. Bowes, D. Gray, and S. Counsell, "A systematic literature review on fault prediction performance in software engineering," *IEEE Trans. Softw. Eng.*, vol. 38, no. 6, pp. 1276–1304, Nov. 2012.

[12] T. Menzies, J. Greenwald, and A. Frank, "Data mining static code attributes to learn defect predictors," *IEEE Trans. Softw. Eng.*, vol. 33, no. 1, pp. 2–13, Jan. 2007.

[13] G. Carrozza, D. Cotroneo, R. Natella, R. Pietrantuono, and S. Russo, "Analysis and prediction of mandelbugs in an industrial software system," in *Proc. IEEE 6th Int. Conf. Softw. Test., Verification Validation*, Mar. 2013, pp. 262–271.

[14] K. El Emam, S. Benlarbi, N. Goel, and S. N. Rai, "Comparing case-based reasoning classifiers for predicting high risk software components," *J. Syst. Softw.*, vol. 55, no. 3, pp. 301–320, Jan. 2001.
[15] M. M. T. Thwin and T.-S. Quah, "Application of neural networks for software quality prediction using object-oriented metrics," *J. Syst. Softw.*, vol. 76, no. 2, pp. 147–156, May 2005.
[16] F. Xing, P. Guo, and M. Lyu, "A novel method for early software quality prediction based on support vector machine," in *Proc. 16th IEEE Int. Symp. Softw. Rel. Eng. (ISSRE)*, Oct. 2006, p. 222.
[17] X. Yuan, T. Khoshgoftaar, E. Allen, and K. Ganesan, "An application of fuzzy clustering to software quality prediction," in *Proc. 3rd IEEE Symp. Appl.-Solidification Syst. Softw. Eng. Technol.*, Nov. 2002, pp. 85–90.
[18] T. M. Khoshgoftaar and N. Seliya, "Software quality classification modeling using the SPRINT decision tree algorithm," *Int. J. Artif. Intell. Tools*, vol. 12, no. 03, pp. 207–225, Sep. 2003.
[19] J. Cahill, J. M. Hogan, and R. Thomas, "Predicting fault-prone software modules with rank sum classification," in *Proc. 22nd Austral. Softw. Eng. Conf.*, Jun. 2013, pp. 211–219.
[20] R. Malhotra, "Comparative analysis of statistical and machine learning methods for predicting faulty modules," *Appl. Soft Comput.*, vol. 21, pp. 286–297, Aug. 2014.
[21] T. M. Khoshgoftaar, Y. Xiao, and K. Gao, "Software quality assessment using a multi-strategy classifier," *Inf. Sci.*, vol. 259, pp. 555–570, Feb. 2014.
[22] S. S. Rathore and S. Kumar, "Towards an ensemble based system for predicting the number of software faults," *Expert Syst. Appl.*, vol. 82, pp. 357–382, Oct. 2017.
[23] E. Erturk and E. Akcapinar Sezer, "Iterative software fault prediction with a hybrid approach," *Appl. Soft Comput.*, vol. 49, pp. 1020–1033, Dec. 2016.
[24] R. Shatnawi, "The application of ROC analysis in threshold identification, data imbalance and metrics selection for software fault prediction," *Innov. Syst. Softw. Eng.*, vol. 13, nos. 2–3, pp. 201–217, Sep. 2017, doi: 10.1007/s11334-017-0295-0.
[25] G. R. Choudhary, S. Kumar, K. Kumar, A. Mishra, and C. Catal, "Empirical analysis of change metrics for software fault prediction," *Comput. Elect. Eng.*, vol. 67, pp. 15–24, Apr. 2018.
[26] D. Sharma and P. Chandra, "Software fault prediction using machine-learning techniques," in *Smart Computing and Informatics*, S. C. Satapathy, V. Bhateja, and S. Das, Eds. Singapore: Springer, 2018, pp. 541–549.
[27] Y. Chen, D. Miao, and R. Wang, "A rough set approach to feature selection based on ant colony optimization," *Pattern Recognit. Lett.*, vol. 31, no. 3, pp. 226–233, Feb. 2010.
[28] M. Charikar, V. Guruswami, R. Kumar, S. Rajagopalan, and A. Sahai, "Combinatorial feature selection problems," in *Proc. 41st Annu. Symp. Found. Comput. Sci.*, Nov. 2002, pp. 631–640.
[29] M. Kudo and J. Sklansky, "Comparison of algorithms that select features for pattern classifiers," *Pattern Recognit.*, vol. 33, no. 1, pp. 25–41, Jan. 2000.
[30] E. Zorarpacı and S. A. Özel, "A hybrid approach of differential evolution and artificial bee colony for feature selection," *Expert Syst. Appl.*, vol. 62, pp. 91–103, Nov. 2016.
[31] H. Zhang and G. Sun, "Feature selection using tabu search method," *Pattern Recognit.*, vol. 35, no. 3, pp. 701–711, Mar. 2002.
[32] M. Mafarja and S. Abdullah, "Fuzzy modified great deluge algorithm for attribute reduction," in *Recent Advances on Soft Computing and Data Mining*, T. Herawan, R. Ghazali, and M. M. Deris, Eds. Cham, Switzerland: Springer, 2014, pp. 195–203.
[33] J. C. W. Debuse and V. J. Rayward-Smith, "Feature subset selection within a simulated annealing data mining algorithm," *J. Intell. Inf. Syst.*, vol. 9, no. 1, pp. 57–81, 1997.
[34] R. Leardi, R. Boggia, and M. Terrile, "Genetic algorithms as a strategy for feature selection," *J. Chemometrics*, vol. 6, no. 5, pp. 267–281, Sep. 1992.
[35] A. A. Heidari, S. Mirjalili, H. Faris, I. Aljarah, M. Mafarja, and H. Chen, "Harris hawks optimization: Algorithm and applications," *Future Gener. Comput. Syst.*, vol. 97, pp. 849–872, Aug. 2019.
[36] H. M. Zawbaa, E. Emary, B. Parv, and M. Sharawi, "Feature selection approach based on moth-flame optimization algorithm," in *Proc. IEEE Congr. Evol. Comput. (CEC)*, Jul. 2016, pp. 4612–4617.
[37] X. Wang, J. Yang, X. Teng, W. Xia, and R. Jensen, "Feature selection based on rough sets and particle swarm optimization," *Pattern Recognit. Lett.*, vol. 28, no. 4, pp. 459–471, Mar. 2007.
[38] M. M. Mafarja and S. Mirjalili, "Hybrid whale optimization algorithm with simulated annealing for feature selection," *Neurocomputing*, vol. 260, pp. 302–312, Oct. 2017.
[39] S. Mirjalili, "Moth-flame optimization algorithm: A novel nature-inspired heuristic paradigm," *Knowl.-Based Syst.*, vol. 89, pp. 228–249, Nov. 2015.
[40] M. Mafarja, I. Aljarah, A. A. Heidari, A. I. Hammouri, H. Faris, A. Al-Zoubi, and S. Mirjalili, "Evolutionary population dynamics and grasshopper optimization approaches for feature selection problems," *Knowl.-Based Syst.*, vol. 145, pp. 25–45, Apr. 2018.
[41] S. Saremi, S. Z. Mirjalili, and S. M. Mirjalili, "Evolutionary population dynamics and grey wolf optimizer," *Neural Comput. Appl.*, vol. 26, no. 5, pp. 1257–1263, Jul. 2015.
[42] J. Kennedy and R. Eberhart, "A discrete binary version of the particle swarm algorithm," in *Proc. IEEE Int. Conf. Syst., Man, Cybern. Comput. Simulation*, vol. 5, Oct. 1997, pp. 4104–4108.
[43] S. Mirjalili and A. Lewis, "S-shaped versus V-shaped transfer functions for binary Particle Swarm Optimization," *Swarm Evol. Comput.*, vol. 9, pp. 1–14, Apr. 2013.
[44] E. Rashedi, H. Nezamabadi-Pour, and S. Saryazdi, "BGSA: Binary gravitational search algorithm," *Natural Comput.*, vol. 9, no. 3, pp. 727–745, Sep. 2010, doi: 10.1007/s11047-009-9175-3.
[45] H. He and E. A. Garcia, "Learning from imbalanced data," *IEEE Trans. Knowl. Data Eng.*, vol. 21, no. 9, pp. 1263–1284, Sep. 2009.
[46] K. W. Bowyer, N. V. Chawla, L. O. Hall, and W. P. Kegelmeyer, "SMOTE: Synthetic minority over-sampling technique," *CoRR*, vol. abs/1106.1813, 2011. [Online]. Available: http://arxiv.org/abs/1106.1813
[47] H. He, Y. Bai, E. A. Garcia, and S. Li, "ADASYN: Adaptive synthetic sampling approach for imbalanced learning," in *Proc. IEEE Int. Joint Conf. Neural Netw. (IEEE World Congr. Comput. Intell.)*, Jun. 2008, pp. 1322–1328.
[48] S. Zhang, X. Li, M. Zong, X. Zhu, and R. Wang, "Efficient kNN classification with different numbers of nearest neighbors," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 29, no. 5, pp. 1774–1785, May 2018.
[49] L. Xu, A. Iosifidis, and M. Gabbouj, "Weighted linear discriminant analysis based on class saliency information," in *Proc. 25th IEEE Int. Conf. Image Process. (ICIP)*, Oct. 2018, pp. 2306–2310.
[50] L. A. Breslow and D. W. Aha, "Simplifying decision trees: A survey," *Knowl. Eng. Rev.*, vol. 12, no. 01, pp. 1–40, Jan. 1997, doi: 10.1017/s0269888997000015.
[51] D. W. Hosmer and S. Lemeshow, *Applied Logistic Regression* (Wiley Series in Probability and Statistics), 2nd ed. Hoboken, NJ, USA: Wiley, 2000.
[52] (2017). *Tera-Promise*. Accessed: Nov. 24, 2017. [Online]. Available: http://openscience.us/repo
[53] M. Jureczko and L. Madeyski, "Towards identifying software project clusters with regard to defect prediction," in *Proc. 6th Int. Conf. Predictive Models Softw. Eng. (PROMISE)*, 2010, pp. 9:1–9:10, doi: 10.1145/1868328.1868342.
[54] A. Okutan and O. T. Yıldız, "Software defect prediction using Bayesian networks," *Empirical Softw. Eng.*, vol. 19, no. 1, pp. 154–181, Feb. 2014.

IYAD TUMAR (Member, IEEE) received the bachelor's degree in electrical engineer (communication systems) and the master's degree in computational science from Birzeit University, Palestine, in 2002 and 2006, respectively, and the Ph.D. degree in smart systems from Jacobs University Bremen, Germany, in 2010. He was a Research Associate and a member of the Computer Networks and Distributed Systems Group, Jacobs University Bremen. He is currently an Assistant Professor with the Electrical and Computer Engineering Department, Birzeit University (BZU). He is also a Project Manager for two research projects funded by Qatar Foundation and BMBF, Germany. His research interests are wireless networks, resource management of disruption tolerant networks, wireless communication, wireless sensor networks, underwater networks, network management, feature selection, software engineering, and machine learning. He was a Research Associate and a Member of European Network of Excellence for the Management of Internet Technologies and Complex Services (EMANICS) from 2007 to 2010. He is a reviewer for IEEE/ACM journals and conferences.

YOUSEF HASSOUNEH is currently an Assistant Professor with the Computer Science Department, Birzeit University, teaching courses in software engineering, Internet programming, and programming languages. He has academic administration experience, as he served as the Department Chair and the Director of the computing master program. He has a profound experience in Human–Computer Interaction, he designed a collaboration framework and groupware tool to enable Requirements Engineering team collaboration. His research interests are in software architecture, virtual software engineering teams, software risk assessment and metrics, and mining software repositories. He has participated in several EU funded projects.

HAMZA TURABIEH received the B.A. and M.Sc. degrees in computer science from Balqa Applied University, Jordan, in 2004 and 2006, respectively, and the Ph.D. degree from the National University of Malaysia (UKM), in 2010. He is currently an Associate professor with the Computer Science Department, Faculty of Science and Information Technology, Taif University. His research interests and activities include the interface of computer science and operational research. Intelligent decision support systems, search and optimization (combinatorial optimization, constraint optimization, multimodal optimization, and multiobjective optimization) using heuristics, local search, hyper-heuristics, met heuristics (in particular memetic algorithms, and particle swarm optimization), and hybrid approaches and their theoretical foundations. Minor interest in machine learning, computational geometry, pattern recognition, image processing, intelligent user-interfaces, and bioinformatics.

THAER THAHER received the B.Sc. degree in computer engineering and the M.Sc. degree in advanced computing from An-Najah National University, Palestine, in 2007 and 2018, respectively. He is currently pursuing the Ph.D. degree in information technology engineering with Arab American University, Palestine Polytechnic University, and Al-Quds University, Palestine. His research interests include evolutionary computation, meta-heuristics, data mining, and machine learning.