Feature Selection Based on Improved Particle Swarm Optimization in Software Defect Prediction

Afnan Alotaibi[1], Nada Alsulaiman[2], Nojoud Albadrani[3], Sultan Alqahtani[4], Wojdan Binsaeedan[5]
Computer Sciences Department, Imam Mohammad Ibn Saud Islamic University
Riyadh, Saudi Arabia
[1]atsalotaibi, [2]nfsulaiman, [3]nfalbadrani {@sm.imamu.edu.sa}, [4]ssalqahtani, [5]wmasaeedan{@imamu.edu.sa}

ABSTRACT

Software testing is an important stage that consumes time, budget, and effort. Thus, software defect prediction helps to identify the defective module, improve the testing process and the overall quality of the software. Many approaches have been conducted by researchers to improve the accuracy of software defect prediction models. Feature selection is a worthy preprocessing technique to enhance applications that uses huge amounts of features. Swarm intelligence algorithms have shown promising performance in improving the feature selection problem and reducing the running time. In this paper, we proposed a software defect prediction model that integrates a sticky binary particle swarm algorithm as feature selection with different classifiers. Also, we compare the performance of the used algorithm with binary particle swarm optimization, and without feature selection. This approach is used for the first time to predict software defect modules, and it provides a promising result where the highest accuracy was obtained from the k-nearest neighbor classifier with the accuracy of 87% using sticky binary particle swarm optimization, 85% using binary particle swarm optimization, and 82% without feature selection.

Keywords: Software defect prediction, Feature selection,Binary particle swarm optimization, swarm intelligence.

1 Introduction

Software Defect Prediction (SDP) is one of the important processes in the software development lifecycle (SDLC) [1]. SDP is the process of identifying and controlling the percentage of defective program units in the software [2] [3]. Defect prediction will improve the testing process and the overall quality of the software [3]. In addition, it will decrease the time, cost, the effort required to rework on the defective units and deliver a more consistent and trusted software [2]. Software defect prediction field started in the 1970s, and now the field have hundreds of related software defect prediction models [4]. In recent studies, SDP models were built using different Machine learning algorithms [5]. The accuracy of SDP models is highly affected by the quality of defect datasets used to build SDP models [6]. The defect dataset features or attributes are growing vastly as software systems are increasing; accordingly, these datasets may contain irrelevant and redundant features that affect the accuracy of classifiers [7] [6].Feature selection (FS) is used to achieve the best classification performance by selecting the most informative feature subset from the dataset [8]. However, FS is an NP-Hard problem because the number of search space increases exponentially when the number of features is increased [7] [9]. Thus, metaheuristic search algorithms are used to find the optimal subset of features within less time [10]. It is random-based methods that generate random solutions and can reach a near-optimal solution [9]. Swarm Intelligence (SI) algorithms are one of the metaheuristic algorithms that are inspired by the behaviours of the creatures such as bees, birds, and fishes [9]. SI algorithms involve two main processes which are exploration and exploitation. Exploration means examine more global regions and find the diverse solution. In contrast, the exploitation is searching locally to enhance the quality of the solution [9]. Over exploration or over exploitation can leads to losses the optimal solution, or premature convergence and suck in local minima [9]. Thus, achieving balance between exploration and exploitation is important for the optimization process. particle swarm optimization (PSO) is one of the SI algorithms that used to solve continuous problems [11]. PSO has been extended to Binary PSO (BPSO) in order to solve discrete problems [12]. This paper introduce SDP model that uses improved FS by combining a sticky binary particle swarm algorithm with classification. In addition to use three

classifiers which are Naive Bayes Classifier (NB), K-Nearest Neighbor Classifier (KNN), and Support Vector Machines (SVM) and compare them. Following are the remaining sections of this paper: Section 2 provides a background on the main related concepts to the SDP, FS, classification algoritms and PSO. Section 3 covers all the related work to this study. Section 4 details the methods utilized to implement the SDP model. Section 5 presents the experimental design, Section 6 presents the results and relevant discussion. Finally, Section 7 concludes the work.

2 Background

This section handles a background on the main related concepts to this study. First, the section provides an overview of SDP process. Then, we give a brief description of FS process and its approaches. In addition, we provide an overview of the used classifiers. Finally, an explaination of the PSO and BPSO.

2.1 Software Defect Prediction (SDP)

The software is exposed to defect during SDLC stages, defect could be arise from the ambiguity of the requirements due to the miscommunication in the requirements elicitation stage, lack of experience in the domain, involving developers with poor technical and programming skills and so forth [3]. Delivering a software with defect will cause huge losses[13]. Customers satisfaction is an important issue for any product, Therefore, to deliver a high quality software, the software has to be achieve customer requirements, and developed within the available budget and time [14]. SDP is important before testing operations take place [3]. Software Quality Assurance (SQA) task consumes (30% - 90%) of the budget of the software project [15]. Therefore, Identifying defective modules and managing the defects in the software will improves the testing process, where it focus on modules that are more expected to work incorrectly [16]. There are three different categories of SDP, prediction the number of defects, the severity of defects and the most frequently used is the prediction of whether the software unit is defective or not [3], where the SDP is used as a binary classification problem that has two classes which are defect and non-defect [17]. Also, it could be used as multi-class classification where the classes are the number of defect.

2.2 Feature Selection (FS)

Feature selection method used to select a subset of the space of features in order to remove the irrelevant features, and to simplify data mining procedures when dealing with large datasets by removing unnecessary and/or duplicate features [6]. Accordingly, FS method enhance the prediction process and the performance of the classifiers [7]. With the advent of improved data collecting techniques, the FS step has become critical and required for improving the mining process' performance. The most difficult aspect of the FS process is determining the ideal feature combination [18]. For datasets with a small number of features, the full (exhaustive) search, which tests all potential combinations before selecting the best one, or the exact search methods can be used. When working with high-dimensional datasets, when the search space is exponentially extended, these search algorithms become impractical and time-consuming. To put it another way, the search space for a dataset with N features consists of (2^N-1) feature subsets [19]. As a result, metaheuristics algorithms are the best alternative to the search techniques stated above.

There are two main approaches to feature subset selection used in machine learning: filter and wrapper. Filter-based approach (FFS) selects subsets of features based on the characteristics of the dataset and exclusively independent of the chosen predictor [6][20]. Accordingly, The advantages of FFS that has a low computational cost and high scalability [6]. Wrapper based approaches (WFS) utilize a prediction model to assess relative usefulness of subsets of features [6][20]. In contrast to the FFS, that asses each feature independently, considering the interdependencies of each feature with the prediction model makes the WFS more WFS accurate [6][21]. Accordingly, this methods are computationally expensive but are more superior to FFS based on existing studies [21][22][23]. In addition to these two main approaches there is a hybrid

technique of both filter and wrapper which is embedded method. In the embedded method, filter method is applied to the initial features to find the best features using some independent criteria; then wrapper method is applied to the same subset and evaluated using the learning algorithm, to find the accuracy [24][2]. Embedded method is less computationally expensive as compared to the wrapper method [24].

2.3 Classification Algorithms

In this study, three classifiers were used to study the effect of the FS on the prediction process and to compare results and measure the improvement. The following subsections provide a brief description of each classifier.

2.3.1 Naive Bayes Classifier (NB)

NB is a probabilistic classifier that based on Bayes' theorem [3].It is assume that there is a strong independence among all the features and they are equally significant [17] [2]. The main advantage of the NB classifier is its short computational time for training [17].

2.3.2 K-Nearest Neighbor Classifier (KNN)

KNN is a classification method that is part of a wider family of pattern recognition algorithmes known as lazy learning or instance-based algorithms [3]. They are based on determining the similarities between the unlabeled new query instance and its nearest k neighbors from the labeled training instance stored in memory rather than performing the generalization in an explicit training phase. The fundamental principle behind k-NN is that close points in space are likely to have similar class concepts. The input to the k-NN in classification problems is the k closest examples among the training examples, and the output is the labels of these instances. For a given example, labeling is determined by the majority of votes cast by the k nearest neighbors. The comparison and computation of the distance between two points are based on a predefined distance metric, such as the Euclidean distance [3].

2.3.3 Support Vector Machines (SVM)

SVM is a supervised learning methods used to analyze data for classification and regression analysis [25]. It was proposed by Vapnik in 1995 [25] . The main idea of SVM is to divides the available data set into two groups by a hyperplane [2] in order to to maximize the distance between the two groups [3]. SVM is a binary classifier that used a linear method in addition to non-linear classification [3].

2.4 Overview of Particle Swarm Optimization (PSO)

PSO is an meta-heurstic technique inroduced by Kennedy and Eberhart [26] that was inspired by social behaviors seen in birds flocking and fish schooling. PSO used to solve many continuous problems [11]. A swarm in PSO is made of multiple individuals known as particles who communicate through iterations to identify optimal solutions while traversing around the search space. PSO performs searches using a population (swarm) of individuals (particles) that are updated from iteration to iteration. The population size is denoted as psize. The solution represented as particle position. To find the optimal solution, each particle modifies its search direction based on two factors: its own best prior experience The solution represented as particle position (pbest) and the best experience of all other particles solutions (gbest).

2.4.1 Binary Particle Swarm Optimization for Feature Selection

To use PSO as feature selection, the position of each particle is represented as a vector where its length is the number of features. Then search for an optimal solution by varying the vector entries values between 0 and 1. Zero means that the feature corresponding to this entry is not selected and one means this feature is selected. Therefore, each particle search for an optimal solution based on the effectiveness of the current solution, and Pbest, and the best solution

from other particles, Gbest. The effectiveness of the solution is measured using the fitness function. The fitness function trains the model on the selected features and evaluates it based on the training.

Many optimization problems in a space are discrete, Binary particle swarm optimization (BPSO) has been proposed to solve these types of tasks [12]. Its another version of PSO that utilize the personal best (pbest) and global best (gbest) solutions to update the velocity and position[27]. The volicity compute how much the particle will move as following [27] [28] :

$$v_i^d(t+1) = wv_i^d(t) + c_1 r_1 \left(pbsest_i^d(t) - x_i^d(t)\right) + c_2 r_2 \left(gbest^d(t) - x_i^d(t)\right) \quad (1)$$

where v is the velocity, w is the inertia weight, c_1 and c_2 are the acceleration factors, r_1 and r_2 are two independent random numbers between 0 and 1, x is the position of particle (solution), $pbsest$ is the personal best solution, $gbsest$ is the global best solution for the population, i is the order of particle in the population, d is the dimension of search space, and t is the number of iterations. All the random numbers will be discussed in section 5.2.

In BPSO, probabilities are used to determine the state of one bit, in other word, a particle moves in a state space constrained to zero and one on each dimension, for example if $v_i^d = 0.20$, then x_i^d will be near to zero more than one [12]. After the velocity is calculated using Equation (1) , it will converted into probability value using Equation (2), is known as a transfer function [29].

$$S\left(v_i^d(t+1)\right) = \frac{1}{1 + \exp\left(-v_i^d\ (t+1)\right)} \quad (2)$$

The position of particle is updated as shown in Equation (3):

$$x_i^d(t+1) = \begin{cases} 1, if\ rand < S\left(v_i^d(t+1)\right) \\ 0, otherwise \end{cases} \quad (3)$$

where $rand$ is a random number between 0 and 1. Considering the minimization function that applied in this paper pbest and gbest is iteratively updated as follows:

$$pbest_i(t+1) = \begin{cases} x_i(t+1), if\ F\left(x_i(t+1)\right) < F(pbest_i(t)) \\ pbest_i(t), otherwise. \end{cases} \quad (4)$$

$$gbest(t+1) = \begin{cases} pbest_i(t+1), if\ F\left(pbest_i(t+1)\right) < F(gbest(t)) \\ gbest(t), otherwise. \end{cases} \quad (5)$$

where x is the solution, F(.) is the fitness function, and t is the number of iterations.

3 Related work

Several research papers in the literature investigate the SDP problem. However, the most recent studies used machine learning approaches to create an appropriate model for this complex challenge. Some studies used pure machine learning methods with no pre-processing on the input datasets [30]. In contrast, others used a combination of pre-processing methods on the datasets, such as feature selection to lower the dimensionality of input datasets or noise reduction from unbalanced datasets. This section will review the works that used the PSO for features selection to improve SDP.

Wahono and Suryana [31] presented a PSO-bagging approach combination to improve the performance of software defect prediction. PSO was employed to deal with feature selection, while the bagging approach was utilized to deal with class imbalance. In order to evaluate the suggested strategy, many machine learning classifiers were applied to nine datasets from NASA's metric data repository. The AUC results showed that the proposed technique enhanced the prediction performance of the most commonly used ML classifiers. Wahono and Ahmad published similar work in [32], in which GA and PSO algorithms were used as feature selection strategies for the SFP problem, while the bagging methodology was used to deal with the class imbalance problem. To test various FS techniques, ten classifiers were used across nine NASA MDP datasets. The AUC results revealed that the proposed FS techniques resulted in a major improvement in prediction performance for the majority of the used classifiers. Furthermore, it has been determined that there is no substantial difference between PSO and GA in feature selection for the majority of classifiers.

Arora and Saha [1] introduced a software defect prediction model based on two classifiers, extreme learning machine (ELM) and kernel-based ELM (KELM). They employed five wrapper-based including PSO and seven filter-based feature selection approaches in their approach. They chose seven datasets from the PROMISE repository to test their method. Furthermore, they employed the accuracy metric to assess the performance feature selection model. They discovered that ELM-based classifiers worked better with wrapper-based feature selection approaches, but KELM classifiers performed better with filter-based methods.

Recently, Malhorta et al. [33] used the Synthetic Minority Oversampling Technique (SMOTE) along with FS using PSO on object-oriented metrics. First, the SMOTE were used to tackle the issue of imbalanced data, while the PSO was applied to extract the optimal feature set. Then, the chosen features were utilized for training the SVM to predict defects. The result showed that the oversampling technique of SMOTE when combined with PSO for feature selection, can be used for building efficient software defect production models.

As shown by past literature, several research publications examine the SDP problem. However, in reality, each project has its characteristics. Thus, it is critical to design a strong model that evaluates the acquired data early in the process. This encourages us to improve the PSO algorithm to use it as a feature selection algorithm as the initial step toward achieving a high-quality classifier.

4 Methodology

This section presents the explanation of the proposed method. The overall system architecture presented in figure 1. The main stages are feature selection then classification. The features selection was used to enhance the model classification by selecting the most important features and remove the noise features. The following subsections explain the system stages.

Figure 1 The System Architecture

4.1 Dataset Specification

We experimented the proposed method on different benchmarked datasets extracted from the PROMISE repository and NASA repository. The datasets were collected from real and open-source software projects. Eleven of these

datasets are downloaded from the NASA corpus[1] (cleaned versions by [34]), while the remaining datasets are from the PROMISE software engineering corpus[2]. These projects have different specifications, such as the programming language, the code size, and software measures, Detailed information on the datasets is listed in Table 1. PROMISE dataset has 20 static attributes and one dependent variable. for each software project. On the other hand, NASA datasets have different number of attributes for each software project. The datasets consist of a set of features that have values and a goal field that describes the instance as defect or non-defect. These features describe the program from different sides including the lines of code measure (program length, count of lines of comments, count of lines of comments), McCabe metrics, base Halstead measures, derived Halstead measures, unique operators, unique operands, total operators, total operands, cyclomatic complexity, essential complexity, design complexity, and a branch-count. We choose PROMISE and NASA because these datasets are commonly used in SDP field. Note that, the datasets used in this study represent the dependent variable by one of two forms: binary form or numric form. In addition, all dataset features are uniformed by deleting the additional columns in dataset other versions files.
The approach for applying training and testing in the experiments is based on a hold-out strategy in which each dataset is shuffled and divided into 70% for training and 30% for testing.

Table 1: *Datasets description.*

	Project	Language	Features	Instances	Defects	Non- Defects	Defect ratio
PROMISE	ant-1.7	Java	21	745	166	579	22.28
	camel-1.4	Java	21	872	145	727	16.63
	camel-1.6	Java	21	965	188	777	19.48
	ivy-2.0	Java	21	352	40	312	11.63
	jedit-4.2	Java	21	367	48	319	13.08
	jedit-4.3	Java	21	492	11	481	2.24
	poi-2.0	Java	21	314	37	277	11.78
	prop-6	Java	21	660	66	594	10.00
	xalan-2.5	Java	21	803	387	416	48.19
	xerces-1.2	Java	21	440	71	369	16.14
NASA	cm1	C	38	327	42	285	12.8
	jm1	C	22	7782	1672	6110	21.5
	kc1	C++	22	1183	314	869	26.5
	kc3	Java	40	194	36	158	18.6
	mc1	C & C++	39	1988	46	1942	2.3
	mw1	C	38	253	27	226	10.7
	pc1	C	38	705	61	644	8.7
	pc2	C	37	745	16	729	2.1
	pc3	C	38	1077	134	943	12.4
	pc4	C	38	1287	177	1110	13.8
	pc5	C++	39	1711	471	1240	27.5

4.2 Feature selection based on Sticky Binary Swarm Optimization

For feature selection, we used the improved binary particle swarm optimization proposed by Nguyen et al. [29] which is Sticky Binary Particle Swarm Optimization (SBPSO). SBPSO works as the flow chart presented in figure 2. The algorithm used the previous position to update the current particle position by flipping the entry d value to zero or one as follows.

$$x_d^{t+1} = \begin{cases} 1 - x_d^t, & \textbf{if } rand() < p_d \\ x_d^t, & \textbf{otherwise}. \end{cases} \quad (6)$$

[1] https://figshare.com/articles/dataset/MDP_data_sets_D_and_D_-_zipped_up/6071675
[2] http://promise.site.uottawa.ca/SERepository/.

Where t represents the iteration number. The SBPSO uses flipping probability p instead of velocity. The flipping probability is expressed as follow:

$$p_d = i_s * (1 - stk_d) + i_p * |pb_d - x_d| + i_g * |gb_d - x_d| \quad (7)$$

i_s , i_p, and i_g are the weights of the sickness, personal best pb_d, and global best gb_d respectively. Where $i_s + i_p + i_g = 1$. The sickness value stk indicates how much the position entry will stick with its value. stk is set to 1 when the entry just flipped then decrease over a fixed number of steps $ustkS$ until the entry flipped or become zero as follow:

$$stk_d^{t+1} = \begin{cases} 1, \textbf{if the bit is just flipped} \\ \max\left(stk_d^t - \frac{1}{ustkS}, 0\right), \textbf{otherwise} \end{cases} \quad (8)$$

They assume that ratio between i_p, and i_g represented as α , $\alpha = i_p / i_g$. The values of i_p, and i_g are expressed as follows:

$$i_p = \alpha \times \frac{1 - i_s}{\alpha + 1} \ , \ i_g = \frac{1 - i_s}{\alpha + 1} \quad (9)$$

Therefore, the flipping probability could be rewrite as the follow:

$$p = \begin{cases} i_s \times (1 - stk_d) & \textbf{if } x_d = pb_d = gb_d \\ i_s \times \left(1 - stk_d - \frac{1}{\alpha + 1}\right) + \frac{1}{\alpha + 1} & \textbf{if } x_d = pb_d \neq gb_d \\ i_s \times \left(1 - stk_d - \frac{\alpha}{\alpha + 1}\right) + \frac{\alpha}{\alpha + 1} & \textbf{if } x_d = gb_d \neq pb_d \\ 1 - i_s * stk_d & \textbf{if } x_d \neq pb_d = gb_d \end{cases} \quad (10)$$

The following paragraphs illustrate each case:

- The first case: $x_d = pb_d = gb_d$ the second and third terms of equation (7) will be zero.
- The second case: $x_d = pb_d \neq gb_d$ the second term in (7) will be zero, and since gb_d and x_d are two different binary numbers the absolute value of their difference is one, hence the third term value is i_g then they substitute it by its value in equation (9).
- The third case: $x_d = gb_d \neq pb_d$ the third term will be zero and the second term is i_p. Then they substitute it by its value in equation (9).
- Last case: $x_d \neq pb_d = gb_d$. The second term is i_p and the third term is i_g. knowing that $i_p + i_g = (1 - i_s)$, the result equation is presented in case 4.

Now, the Flipping probability depends on two parameters stickiness weight i_s and $ustkS$ that control the sickness. α is a fixed value that is set before starting the algorithm. i_s and $ustkS$ are dynamically updated in each iteration as follow:

$$ustkS^t = ustkS^L + \frac{t}{T} * (ustkS^U - ustkS^L) \quad (11)$$

$$i_s^t = i_s^U - \frac{t}{T} * (i_s^U - i_s^L) \quad (12)$$

Where T is the total number of iterations, t is the current iteration U is upper bound, L is lower bound. The fitness function that assesses the position of the particle is the model error rate computed as the following:

$$Error_{rate} = \frac{flase\ prediction}{total\ number\ of\ sampe} \quad (13)$$

False prediction is the incorrect prediction on the testing dataset using the selected features that correspond to the particle's position. The total number of samples is the length of the testing dataset.

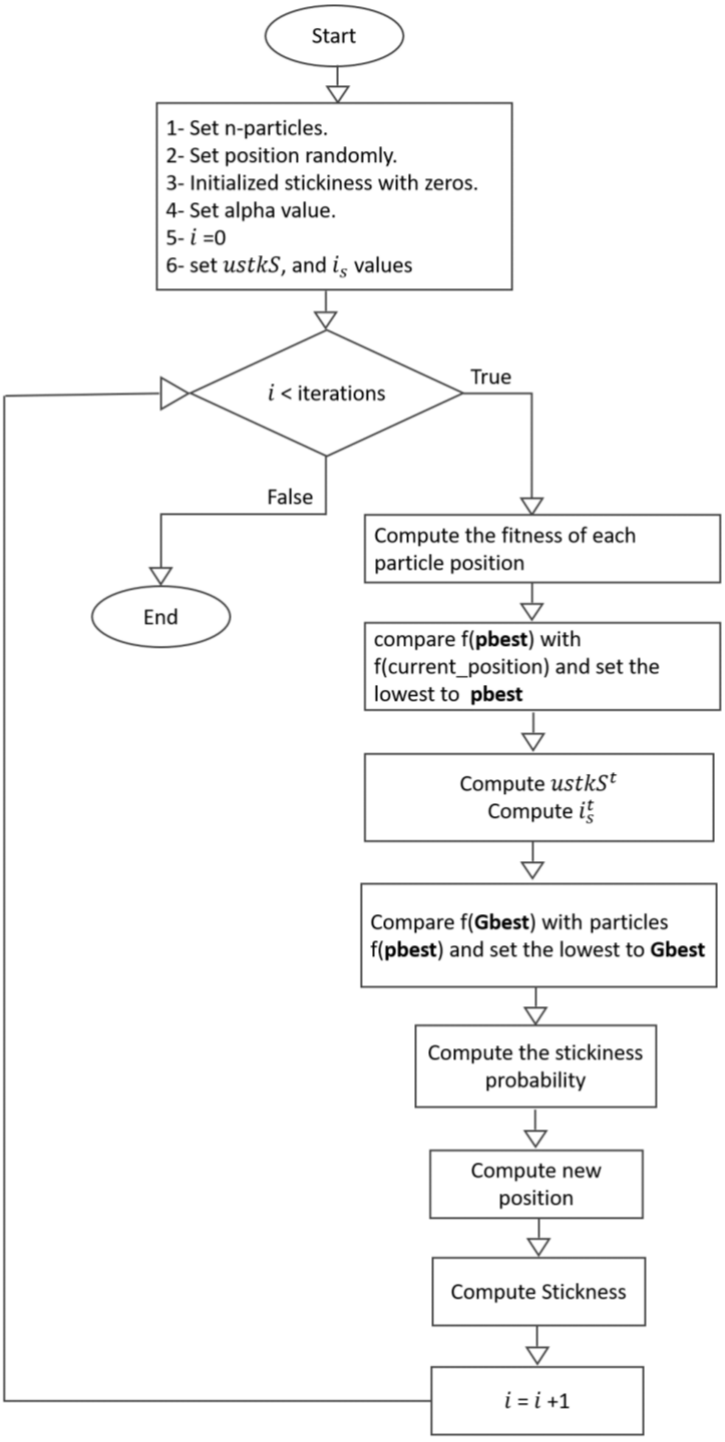

Figure 2 SBPSO flowchart.

4.3 Classification

For classification, we train three machine learning models, SVM, KNN, and NB on the datasets. The three models were trained on the selected features from the previous stage. Also, we compare the models trained on selected features from SBPSO, BPSO, and without feature selection.

4.4 Evaluation

In the evaluation stage, the models were evaluated by predicting the defect using the testing dataset then we compare the prediction with the true label in the dataset. The assessment of each model is based on the evaluation metrics explained in section 5.2.

5 Experimental Design

Both BPSO and SBPS have been implemented using the pyswarm[3] library using Python programming language. The following subsections explain the parameters setting and evaluations metrics used in this work.

5.1 Parameter setting

The BPSO parameters are w, c_1, c_2, numbers of particles, numbers of iterations. After testing ranges of values, the selected values are $w = 0.4, c_1 = 0.6, c_2 = 0.3$. Regarding SBPSO its parameters are α, lower and upper bound of i_s and $ustkS$, Numbers of particles and numbers of iterations. α set to 0.6, lower bound of $i_s = 0$ and upper bound of $i_s = \frac{10}{n}$ where n is the number of features. Lower bound of $ustkS = 1 * \frac{T}{100}$ and upper bound of $i_s = 1 * \frac{T}{100}$, where T is the total number of iterations. The numbers of particles and numbers of iterations are the same for both BPSO and SBPSO 20 and 100 respectively.

5.2 Model Evaluation Metrics

We employed a binary classification strategy to identify classes that are likely to have defects. A binary classifier can make two types of errors: false positives (FP) and false negatives (FN). A correctly categorized defected class is also a true positive (TP), whereas a correctly classified non-defected class is a true negative (TN). We examined classification outcomes using the following metrics: Accuracy, Precision, Recall, and F-measure, described as follows:

- **Accuracy:** address the number or percentage of correctly classified instances to the total sum of instances. It can be calculated by Equation (14):

$$Accuracy = \frac{TP + TN}{TP + FP + FN + TN} \quad (14)$$

- **Precision:** Addresses the percentage of successfully classified defective occurrences among the total number of retrieved instances. The best precision value is 1. The higher the precision is, the fewer false positives. It can be calculated by Equation (15):

$$Precision = \frac{TP}{TP + FP} \quad (15)$$

- **Recall:** Addresses The percentage of defective cases that were accurately classified. The best recall value is 1. The higher the recall is, the lower the number of false negatives. It can be calculated by Equation (16):

$$Recall = \frac{TP}{TP + FN} \quad (16)$$

[3] https://pyswarms.readthedocs.io/en/latest/#

- **F-measure:** computes accuracy by taking Precision and Recall into account, which may be understood as a weighted average of Precision and Recall. The F-measure value varies between 0 and 1, with values closer to 1 indicating better classification results. It can be calculated by Equation (17):

$$F - measure = 2 \frac{Recall * Precision}{Recall + Precision} \quad (17)$$

6 Results and Discussion

In this section, the three classifiers results are compared which are NB, KNN, and SVM through three experiment for each classifier: without FS, with BPSO, and with the proposed SBPSO. The figures 3-9 shows the visulazation resuls of each metric. Also, Table 2 shows the numerical results of each experiment.

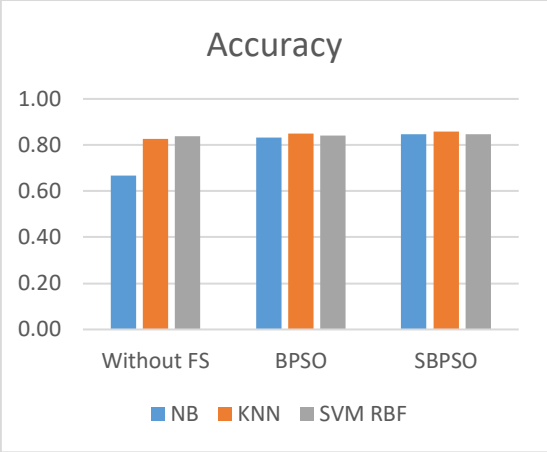

Figure 3 Average accuracy of the three classifiers

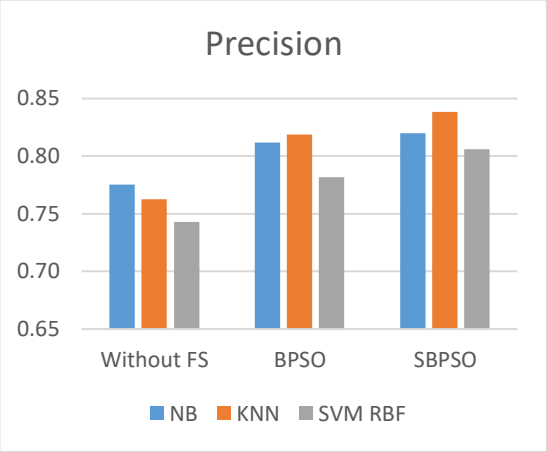

Figure 4 Average precision of the three classifiers

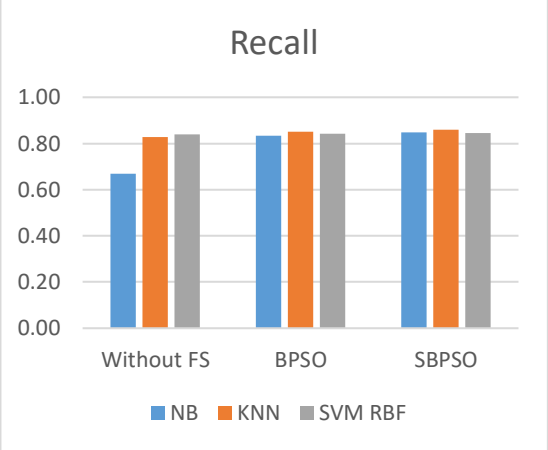

Figure 5 Average recall of the three classifiers

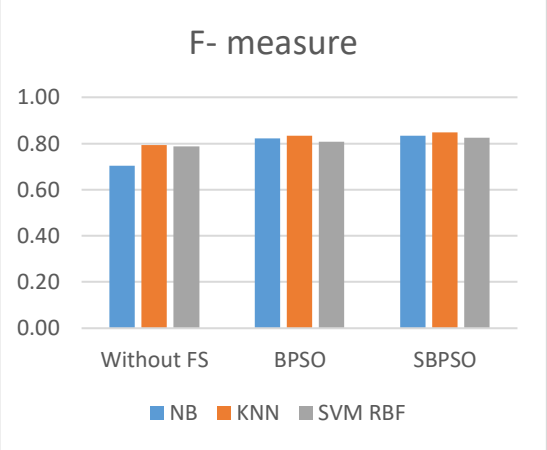

Figure 6 Average F-measure of the three classifiers

6.1 SBPSO effectivity

As can be seen figure 3 shows the average accuracy obtained from testing the classifiers NB, KNN, and SVM. We observe that using FS with classification improve the accuracy. The lowest accurcies from the three classifiers were obtained when the classifiers trained and tested on the datasets without FS. There was an increase in the accuracy values of the three classifiers when use BPSO as FS. The best accuracy results were obtained when use the SBPSO in FS stage. This can be explained by the using FS stage that have an

effective influence on the classifiers' performance. Moreover, figure 4, clearly shows that precision was improved after using SBPSO. Using SBPSO also slightly improved the Recall and F-measure as shown in figure 5 and 6 respectively.

6.2 Exploration and Exploitation of SBPSO and BPSO

Exploration and exploitation are important factors for searching for optimal solutions. Exploration is searching in the whole search space to visit all locations, while exploitation is searching between visited locations [35]. Updating the positions in SBPSO depend on the previous position and the probability of flipping (as equation (6) shows). If the probability is high the entry will flip. The flipping probability depends on the stickiness value of the entry, if the stickiness is high the flipping probability is low (as equation (7) show). The stickiness strategy balances between exploration and exploitation. The entry stick in the current value and other entries will search around for a fixed number of steps $ustkS$ (exploitation) then the entry will flip (exploration). The results from table. 2 show that SBPSO outperforms BPSO. This is due to the optimal features' subset that SBPSO selects from its $gbest$. BPSO changes the position entry based on the velocity that computes how far the particle will go depending on $gbest$ and $pbest$ location (3).

6.3 Difference between classifiers

Figure 3 shows that the SVM classifier was the best in the experiment that was done without FS, while in the BPSO and SBPSO experiments, **KNN** outperformed the rest of the classifiers followed by the SVM. As we observe from figures 3-8, the classifier which is the most influential by features selection is NB we think that due to its mechanism where it gives each feature a Bayesian probability. The probability is affected by the number of features. Therefore, by removing unnecessary features the probability will more accurate and the influential features on the classification will have a higher probability.

Table 2: *Average results of the three classifiers NB, KNN, and SVM RBF*

		NB			KNN			SVM RBF		
		Without FS	BPSO	SBPSO	Without FS	BPSO	SBPSO	Without FS	BPSO	SBPSO
PROMICE	Acuracy	0.60	0.82	0.84	0.83	0.85	0.85	0.83	0.83	0.84
	Precision	0.74	0.79	0.80	0.76	0.79	0.81	0.71	0.73	0.78
	Recall	0.60	0.82	0.84	0.83	0.85	0.85	0.83	0.83	0.84
	F-measure	0.65	0.81	0.82	0.79	0.82	0.83	0.77	0.78	0.81
NASA	Acuracy	0.73	0.84	0.85	0.82	0.85	**0.87**	0.85	0.85	0.85
	Precision	0.80	0.83	0.84	0.77	0.84	**0.87**	0.77	0.83	0.83
	Recall	0.75	0.83	0.85	0.80	0.85	**0.87**	0.81	0.84	0.84
	F-measure	0.67	0.80	0.81	0.78	0.81	**0.82**	0.76	0.77	0.80

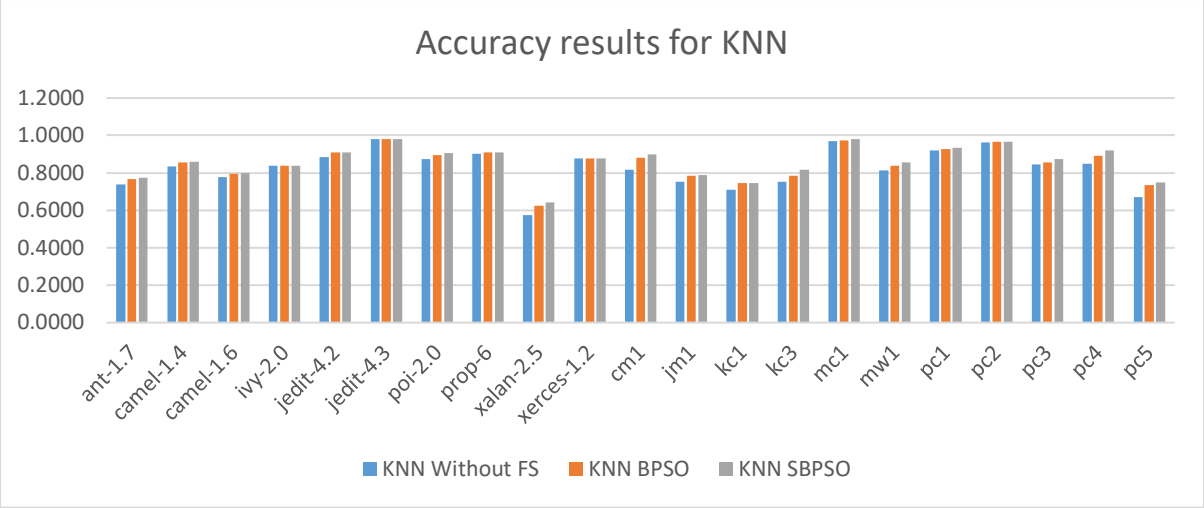

Figure 7 Accuracy results for KNN

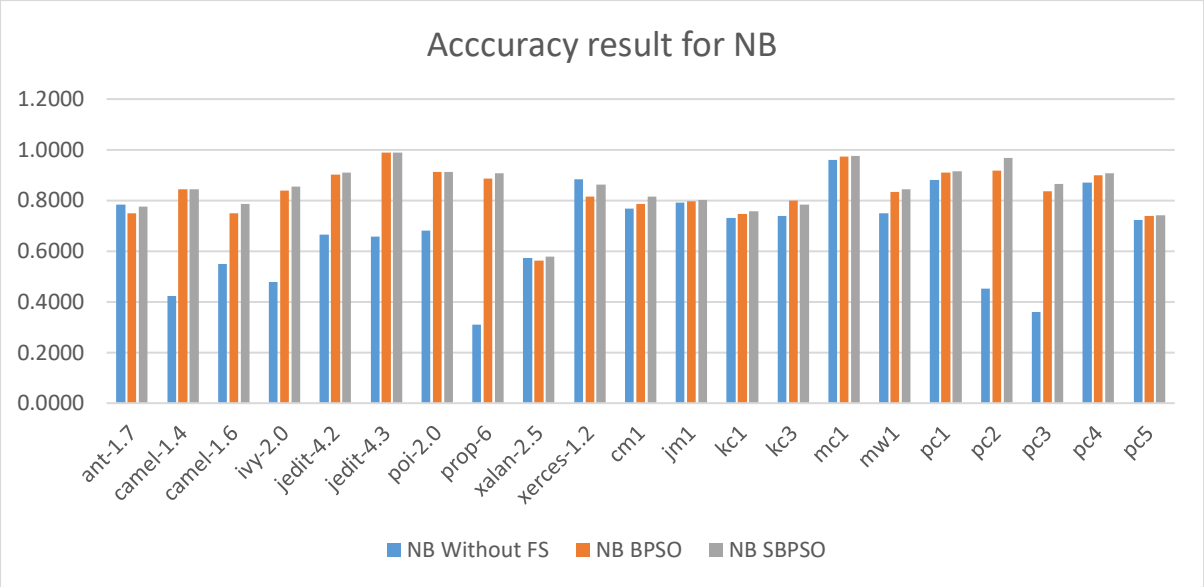

Figure 8 Accuracy results for NB

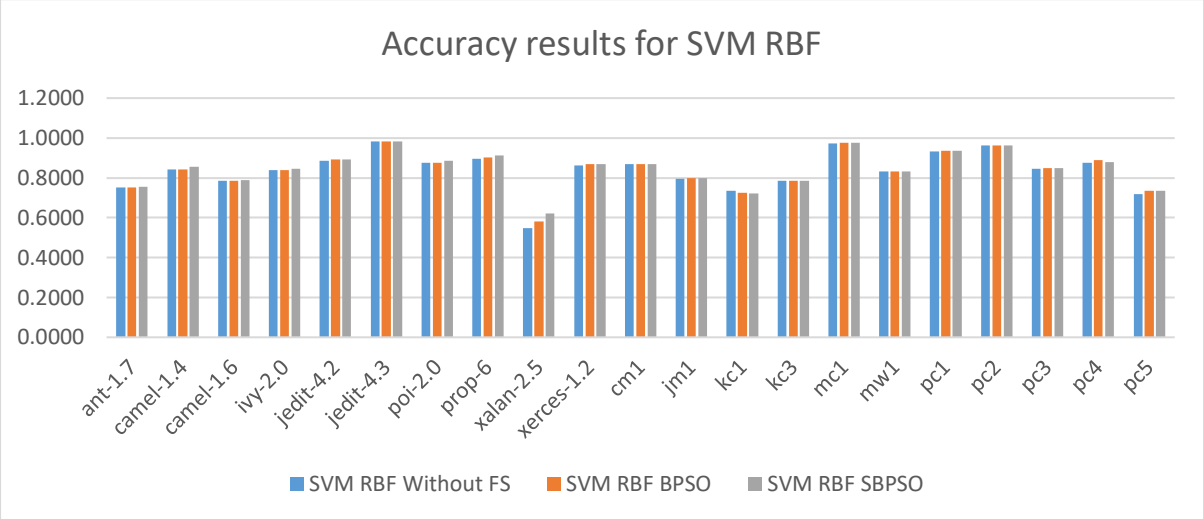

Figure 9 Accuracy results for SVM RBF

6.4 Analytical Description of the Selected Features

As [3] do in this section we present the selected features by SBPSO for each dataset. Table 3 present the selected features from each dataset, the number of All features (AF), Selected Features (SF) and Features Reduction Ratio (FRR). The FRR is computed as the following:

$$FRR = \frac{AF - SF}{Af} \tag{18}$$

We compare the influence of features for the PROMISE dataset since the features for each dataset are the same. The most selected features in PROMISE are F10 and F12. They selected 8 times in 10 datasets with a selection ratio of 0.8. Table 4 present the feature number, number of selected times, selection ratio, and the feature.

Table 3: *Selected features in software datasets.*

Project	AF	SF	FFR%	Selected Features
ant-1.7	21	10	52	F2, F3, F4, F5, F8, F9, F12, F14, F15, F20
camel-1.4	21	8	62	F3, F5, F8, F10, F12, F13, F16, F19
camel-1.6	21	9	57	F1, F4, F7, F8, F12, F14, F15, F18, F20
ivy-2.0	21	8	62	F1, F3, F4, F7, F9, F10, F14, F17
jedit-4.2	21	12	43	F1, F3, F4, F5, F8, F9, F10, F13, F14, F17, F18, F20
jedit-4.3	21	10	52	F7, F8, F10, F12, F13, F15, F17, F18, F19, F20
poi-2.0	21	8	62	F2, F10, F12, F15, F16, F18, F19, F20
prop-6	21	11	48	F3, F4, F5, F6, F8, F10, F12, F14, F16, F17, F19
xalan-2.5	21	10	52	F1, F8, F10 F12 F14, F15, F18, F20
xerces-1.2	21	9	57	F1, F3, F4, F6, F9, F10, F11, F12, F16
cm1	38	14	63	F1, F4, F6, F7, F9, F14, F15, F22,F23, F26, F31, F32, F3, F35
jm1	22	12	45	F2, F3, F4, F8, F9, F10, F13, F16, F17, F18, F19, F20
kc1	22	7	68	F2, F4, F5, F14, F17, F18, F20
kc3	40	19	53	F3, F6, F7, F9, F10, F11, F12, F15, F18, F19, F24, F25, F28, F31, F33, F34, F35, F37, F38
mc1	39	17	56	F1, F2, F3, F4, F5, F6, F9, F10, F12, F14, F17, F20, F22, F24, F29, F30, F37
mw1	38	17	55	F1, F5, F6, F7, F8, F11, F12, F14, F16, F17, F21, F25, F26, F29, F31, F32, F35
pc1	38	14	63	F2, F3, F4, F7, F9, F10, F12, F17, F19, F28, F29, F31, F32, F36
pc2	37	15	59	F3, F4, F5, F6, F9, F12, F15, F16, F20, F21, F24, F25, F29, F30, F31
pc3	38	17	55	F1, F2, F3, F7, F11, F15 , F18, F22, F23, F24, F26, F28, F29, F32, F34, F35, F36
pc4	38	10	74	F3, F9, F10, F12 , F15, F22, F24, F26, F30, F32
pc5	39	13	67	F1,F2, F3, F4, F15, F17, F18, F27, F31, F33, F34, F36, F37

Table 4: *PROMISE dataset features selection*

F no.	Feature description	Selected times	selection ratio
F1	Weighted methods per class	5	0.25
F2	Depth of inheritance tree	2	0.1
F3	Number of children	5	0.25

F4	Coupling between objects	6	0.3
F5	Response for a class	4	0.2
F6	Lack of cohesion in methods	2	0.1
F7	Another lcom metric proposed by Henderson-Sellers	3	0.15
F8	Number of public methods	7	0.35
F9	Number of lines of code	4	0.2
F10	Fraction of private or protected attributes	**8**	**0.4**
F11	Number of fields that are user-defined types	1	0.05
F12	Fraction of accessible methods that are inherited	**8**	**0.4**
F13	Cohesion among methods of a class based on parameter list	3	0.15
F14	Inheritance coupling	6	0.3
F15	Coupling between methods	5	0.25
F16	Average method complexity	3	0.15
F17	Number of classes depending on a class	4	0.2
F18	Number of classes a class depends on	5	0.25
F19	Maximum McCabe's cyclomatic complexity score of methods	3	0.15
F20	Mean of McCabe's cyclomatic complexity score of methods	5	0.25

7 Conclusions

In conclusion, in this research, we developed the SDP system that predicts if the software has a defect or not based on a set of features, we use the FS algorithm SBPSO to select the most important feature. We compare the SDP performance using SBPSO as FS, BSPSO as FS, and without FS. Also, we conducted experiments on three classifiers, KNN, SVM, NB, and on two datasets, Promice and Nasa. SDP using SBPSO outperform BPSO and SDP without FS. Moreover, we discuss the effectiveness of SBPSO and compare it with BSPO in terms of exploration and exploitation. SBPSO balances between exploration and exploitation using the stickiness strategy. We show that the NB classifier is the most affected classifier by feature selection. In addition, we analyze the most selected features.

References

[1] I. Arora and A. Saha, "ELM and KELM based software defect prediction using feature selection techniques," *Journal of Information and Optimization Sciences*, vol. 40, no. 5, pp. 1025–1045, Jul. 2019, doi: 10.1080/02522667.2019.1637999.

[2] M. Anbu and G. S. Anandha Mala, "Feature selection using firefly algorithm in software defect prediction," *Cluster Comput*, vol. 22, no. 5, pp. 10925–10934, Sep. 2019, doi: 10.1007/s10586-017-1235-3.

[3] R. A. Khurma, H. Alsawalqah, I. Aljarah, M. A. Elaziz, and R. Damaševičius, "An Enhanced Evolutionary Software Defect Prediction Method Using Island Moth Flame Optimization," *Mathematics*, vol. 9, no. 15, Art. no. 15, Jan. 2021, doi: 10.3390/math9151722.

[4] W. Bi and F. Yu, "A Feature Selection Framework for Software Defect Prediction Using ISFLA," *IOP Conf. Ser.: Mater. Sci. Eng.*, vol. 677, p. 052121, Dec. 2019, doi: 10.1088/1757-899X/677/5/052121.

[5] D. Bowes, T. Hall, and J. Petrić, "Software defect prediction: do different classifiers find the same defects?," *Software Quality Journal*, vol. 26, Jun. 2018, doi: 10.1007/s11219-016-9353-3.

[6] A. O. Balogun *et al.*, "Search-Based Wrapper Feature Selection Methods in Software Defect Prediction: An Empirical Analysis," *Advances in Intelligent Systems and Computing*, vol. 1224 A, pp. 492–503, 2020.

[7] S. Goyal and P. K. Bhatia, "Software fault prediction using lion optimization algorithm," *Int. j. inf. tecnol.*, Sep. 2021, doi: 10.1007/s41870-021-00804-w.

[8] M. Kondo, C. P. Bezemer, Y. Kamei, A. E. Hassan, and O. Mizuno, "The impact of feature reduction techniques on defect prediction models," *Empirical Software Engineering*, vol. 24, no. 4, pp. 1925–1963, Aug. 2019, doi: 10.1007/s10664-018-9679-5.

[9] R. Abu Khurma, I. Aljarah, and A. Sharieh, "A Simultaneous Moth Flame Optimizer Feature Selection Approach Based on Levy Flight and Selection Operators for Medical Diagnosis," *ARABIAN JOURNAL FOR SCIENCE AND ENGINEERING*, Mar. 2021, doi: 10.1007/s13369-021-05478-x.

[10] S. Hosseini, B. Turhan, and M. Mäntylä, "A benchmark study on the effectiveness of search-based data selection and feature selection for cross project defect prediction," *Information and Software Technology*, vol. 95, pp. 296–312, Mar. 2018, doi: 10.1016/j.infsof.2017.06.004.

[11] Y.-J. Gong *et al.*, "Genetic Learning Particle Swarm Optimization," *IEEE transactions on cybernetics*, vol. 46, Sep. 2015, doi: 10.1109/TCYB.2015.2475174.

[12] J. Kennedy and R. C. Eberhart, "A discrete binary version of the particle swarm algorithm," in *Computational Cybernetics and Simulation 1997 IEEE International Conference on Systems, Man, and Cybernetics*, Oct. 1997, vol. 5, pp. 4104–4108 vol.5. doi: 10.1109/ICSMC.1997.637339.

[13] D. R. Ibrahim, R. Ghnemat, and A. Hudaib, "Software Defect Prediction using Feature Selection and Random Forest Algorithm," in *2017 International Conference on New Trends in Computing Sciences (ICTCS)*, Oct. 2017, pp. 252–257. doi: 10.1109/ICTCS.2017.39.

[14] D. Harekal and V. Suma, "Implication of Post Production Defects in Software Industries," *International Journal of Computer Applications*, vol. 109, pp. 20–23, Jan. 2015, doi: 10.5120/19419-1032.

[15] M. Kläs, H. Nakao, F. Elberzhager, and J. Münch, "Support planning and controlling of early quality assurance by combining expert judgment and defect data- A case study," *Empirical Software Engineering*, vol. 15, pp. 423–454, Aug. 2010, doi: 10.1007/s10664-009-9112-1.

[16] N. D. Singpurwalla, "Determining an Optimal Time Interval for Testing and Debugging Software," *IEEE TRANS SOFTWARE ENG*, vol. 17, no. 4, pp. 313–319, Apr. 1991, doi: 10.1109/32.90431.

[17] R. Wahono, "A Systematic Literature Review of Software Defect Prediction: Research Trends, Datasets, Methods and Frameworks," *Journal of Software Engineering*, vol. 1, May 2015.

[18] M. Mafarja *et al.*, "Binary dragonfly optimization for feature selection using time-varying transfer functions," *Knowledge-Based Systems*, vol. 161, pp. 185–204, Dec. 2018, doi: 10.1016/j.knosys.2018.08.003.

[19] T. Thaher and N. Arman, "Efficient Multi-Swarm Binary Harris Hawks Optimization as a Feature Selection Approach for Software Fault Prediction," in *2020 11th International Conference on Information and Communication Systems (ICICS)*, Apr. 2020, pp. 249–254. doi: 10.1109/ICICS49469.2020.239557.

[20] I. Guyon and A. Elisseeff, "An Introduction to Variable and Feature Selection," *Journal of Machine Learning Research*, vol. 3, no. Mar, pp. 1157–1182, 2003.

[21] M. A., A. Balogun, H. Mojeed, and E. Ayobami, "Wrapper Feature Selection based Heterogeneous Classifiers for Software Defect Prediction," pp. 1–11, Feb. 2019.

[22] Z. Xu, J. Liu, Z. Yang, G. An, and X. Jia, "The Impact of Feature Selection on Defect Prediction Performance: An Empirical Comparison," in *2016 IEEE 27th International Symposium on Software Reliability Engineering (ISSRE)*, Oct. 2016, pp. 309–320. doi: 10.1109/ISSRE.2016.13.

[23] B. Ghotra, S. McIntosh, and A. E. Hassan, "A Large-Scale Study of the Impact of Feature Selection Techniques on Defect Classification Models," in *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*, May 2017, pp. 146–157. doi: 10.1109/MSR.2017.18.

[24] S. Biswas, M. Bordoloi, and B. Purkayastha, "Review on Feature Selection and Classification using Neuro-Fuzzy Approaches," *International Journal of Applied Evolutionary Computation*, vol. 7, pp. 28–44, Oct. 2016, doi: 10.4018/IJAEC.2016100102.

[25] H. Can, X. Jianchun, Z. Ruide, L. Juelong, Y. Qiliang, and X. Liqiang, "A new model for software defect prediction using Particle Swarm Optimization and support vector machine," in *2013 25th Chinese Control and Decision Conference (CCDC)*, May 2013, pp. 4106–4110. doi: 10.1109/CCDC.2013.6561670.

[26] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proceedings of ICNN'95 - International Conference on Neural Networks*, Perth, WA, Australia, 1995, vol. 4, pp. 1942–1948. doi: 10.1109/ICNN.1995.488968.

[27] J. Too, A. R. Abdullah, and N. Mohd Saad, "A New Co-Evolution Binary Particle Swarm Optimization with Multiple Inertia Weight Strategy for Feature Selection," *Informatics*, vol. 6, no. 2, Art. no. 2, Jun. 2019, doi: 10.3390/informatics6020021.

[28] L.-Y. Chuang, H.-W. Chang, C.-J. Tu, and C.-H. Yang, "Improved binary PSO for feature selection using gene expression data," *Comput Biol Chem*, vol. 32, no. 1, pp. 29–37, Feb. 2008, doi: 10.1016/j.compbiolchem.2007.09.005.

[29] B. Nguyen, B. Xue, P. Andreae, and M. Zhang, "A New Binary Particle Swarm Optimization Approach: Momentum and Dynamic Balance Between Exploration and Exploitation," *IEEE Transactions on Cybernetics*, vol. PP, pp. 1–15, Oct. 2019, doi: 10.1109/TCYB.2019.2944141.

[30] E. Erturk and E. A. Sezer, "A comparison of some soft computing methods for software fault prediction," *Expert Systems with Applications*, vol. 42, no. 4, pp. 1872–1879, Mar. 2015, doi: 10.1016/j.eswa.2014.10.025.

[31] R. S. Wahono and N. Suryana, "Combining Particle Swarm Optimization based Feature Selection and Bagging Technique for Software Defect Prediction," *International Journal of Software Engineering and Its Applications*, p. 14, 2013.

[32] R. Wahono, N. Suryana, and S. Ahmad, "Metaheuristic Optimization based Feature Selection for Software Defect Prediction," *Journal of Software*, vol. 9, pp. 1324–1333, May 2014, doi: 10.4304/jsw.9.5.1324-1333.

[33] R. Malhotra, N. Nishant, S. Gurha, and V. Rathi, "Application of Particle Swarm Optimization for Software Defect Prediction Using Object Oriented Metrics," in *2021 11th International Conference on Cloud Computing, Data Science Engineering (Confluence)*, Jan. 2021, pp. 88–93. doi: 10.1109/Confluence51648.2021.9377116.

[34] M. Shepperd, Q. Song, Z. Sun, and C. Mair, "Data Quality: Some Comments on the NASA Software Defect Datasets," *IIEEE Trans. Software Eng.*, vol. 39, no. 9, pp. 1208–1215, Sep. 2013, doi: 10.1109/TSE.2013.11.

[35] M. N. M. Salleh *et al.*, "Exploration and Exploitation Measurement in Swarm-Based Metaheuristic Algorithms: An Empirical Analysis," in *Recent Advances on Soft Computing and Data Mining*, vol. 700, R. Ghazali, M. M. Deris, N. M. Nawi, and J. H. Abawajy, Eds. Cham: Springer International Publishing, 2018, pp. 24–32. doi: 10.1007/978-3-319-72550-5_3.